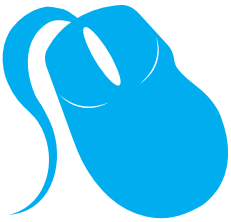
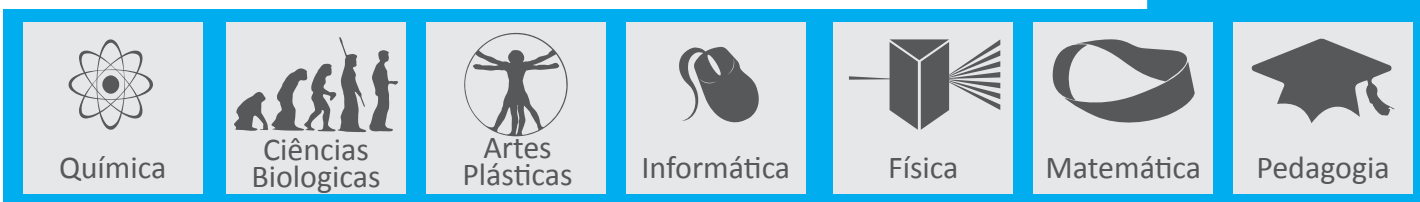


Informática

Fundamentos de Engenharia de Software

Mariela Inés Cortés

Fortaleza
2013



Presidente da República
Dilma Vana Rousseff

Ministro da Educação
Henrique Paim Fernandes

Presidente da CAPES
Jorge Almeida Guimarães

Diretor de Educação a Distância da CAPES
Jean Marc

Governador do Estado do Ceará
Cid Ferreira Gomes

Reitor da Universidade Estadual do Ceará
José Jackson Coelho Sampaio

Pró-Reitora de Graduação
Marcília Chagas Barreto

Coordenador da SATE e UAB/UECE
Francisco Fábio Castelo Branco

Coordenadora Adjunta UAB/UECE
Eloísa Maia Vidal

Diretor do CCT/UECE
Luciano Moura Cavalcante

Coordenação da Licenciatura em Informática
Francisco Assis Amaral Bastos

Coordenação de Tutoria da Licenciatura em Informática
Maria Wilda Fernandes Felipe

Coordenadora Editorial
Rocylânia Isidio de Oliveira

Projeto Gráfico e Capa
Roberto Santos

Diagramador
Francisco Oliveira

Secretaria de Apoio às Tecnologias Educacionais
Av. Paranjana, 1700 - Campus do Itaperi - Fortaleza - Ceará
(85) 3101-9962

Sumário

Apresentação

Na constante busca por software de qualidade desenvolvido levando em conta um equilíbrio satisfatório entre o custo investido e o beneficio propiciado, surge uma área da Ciência da Computação: a Engenharia de Software. A abordagem de engenharia à construção de software traz como beneficio a sistematização das atividades de planejamento e construção.

Independentemente do domínio da aplicação sendo desenvolvida, a Engenharia de Software surge como pilar fundamental para a construção de software de qualidade, desenvolvido levando em conta restrições sobre o tempo, orçamento e escopo objetivando a satisfação do cliente.

O presente livro apresenta de forma clara e amigável os princípios, métodos e práticas que guiam a Engenharia de Software e que vêem sendo adotados com sucesso não somente no âmbito acadêmico. Organizações e indústrias nos mais diversos segmentos de atuação estão cada vez mais preocupadas na adoção de métodos e modelos de processo como um meio de alcançar maior competitividade no mercado.

O livro está organizado em quatro unidades. A primeira unidade fornece as bases necessárias para o entendimento dos conceitos básicos e premissas que guiam as atividades que fazem parte do ciclo de vida do desenvolvimento de software. Na Unidade 2 são apresentados os fluxos de atividades que normalmente participam do processo de desenvolvimento de software. Adicionalmente, encadeamentos típicos destes fluxos através dos modelos de ciclo de vida comumente utilizados para o desenvolvimento de software são apresentados. A Unidade 3 tem seu foco no conceito de qualidade e sua garantia em projetos. A Unidade 4 apresenta as metodologias, técnicas e ferramentas mais amplamente utilizadas para o desenvolvimento de software: o método estruturado e orientado a objetos.

O conteúdo apresentado no presente livro destina-se principalmente a professores e alunos de graduação em Ciências da Computação ou áreas afins, fornecendo um embasamento teórico e uma clara noção dos princípios e estratégias a serem seguidos no desenvolvimento de software de qualidade, com foco na plena satisfação do cliente.

A Autora

Capítulo 1

Visão geral da Engenharia de Software

Objetivos

- Os fundamentos da Engenharia de Software advindos da engenharia tradicional visam o desenvolvimento de um produto adequado às necessidades e expectativas do cliente. Nesta unidade são apresentados os conceitos básicos, princípios e métodos da Engenharia de Software que orientam o desenvolvimento de produtos de software nos diversos domínios de aplicação.

1. Introdução

Há décadas, aplicações de software participam da realização e controle de praticamente todas as atividades na nossa volta. O inesgotável avanço da tecnologia e o aumento constante da exigência dos usuários fazem com que sistemas computacionais se tornem cada vez mais complexos.

Neste contexto, o desenvolvimento e construção de sistemas que atendam a estas demandas se tornam um desafio cada vez mais difícil de ser alcançado. A Engenharia de Software é uma área da Ciência da Computação relativamente nova, que surgiu para auxiliar neste processo, visando o desenvolvimento de produtos de software com uma boa relação entre custo e beneficio.

Engenharia (do latim ingeniu = "faculdade inventiva, talento") é a arte, a ciência e a técnica de bem conjugar os conhecimentos especializados (científicos) de uma dada área do saber com a sua viabilidade técnico-econômica, para produzir novas utilidades e/ou transformar a natureza, em conformidade com idéias bem planejadas.

A Engenharia de Software pode ser definida como uma disciplina que utiliza um conjunto de métodos, técnicas e ferramentas para analisar, projetar e gerenciar o desenvolvimento e manutenção de software, visando produzir e manter software dentro de prazos, custos e qualidade estimados.

Estes objetivos são alcançados através de uma abordagem sistemática para o desenvolvimento, operação e descarte de software, aplicando a prática de conhecimento científico ao projeto e construção de software. Esta abordagem tem sua origem nos princípios propiciados pela engenharia tradicional.

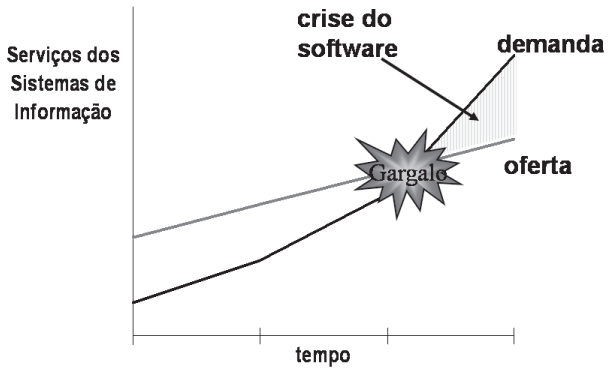
A Engenharia de Software objetiva o aumento da qualidade do **software**, visando paralelamente uma maior produtividade da equipe envolvida nas atividades de desenvolvimento e atingindo uma maior satisfação por parte dos clientes.

O software envolve não somente o conjunto de programas ou código fonte da aplicação, mas também toda documentação associada gerada durante o processo de desenvolvimento.

A Engenharia de Software surgiu no contexto da chamada **Crise do Software** quando, a partir do avanço tecnológico e a consequente redução de custos na aquisição de computadores domésticos, houve um aumento na demanda de software para um mercado profissional que não estava preparado para atender a essa nova realidade.

Esta crise teve lugar na década dos 70´s, e foi constatada por um gargalo no processo de desenvolvimento de software, onde a oferta de mão de obra capacitada não conseguia atender à demanda. Algumas consequências deste processo foram:

- Custos elevados;
- Atrasos na entrega;
- Baixa confiabilidade e corretude do produto;
- Dificuldade de medição;
- Gerência do desenvolvimento ineficaz;
- Alta demanda não atendida;
- Difícil manutenção do software existente.



De forma geral, uma maior ênfase era dada à programação, deixando de lado a adoção de princípios mais fundamentais, ou boas práticas que possibilitassem um melhor gerenciamento da complexidade no desenvolvimento de tais sistemas.

A carência de uma sistemática no processo de desenvolvimento de software é caracterizada pela utilização das denominadas práticas artesanais para o desenvolvimento. O uso de práticas artesanais pode ser adequada em projetos menores e de curto prazo, que envolvem um domínio bem conhecido e com um número reduzido de envolvidos. No entanto, de forma geral, a falta de uma definição precisa do processo a ser seguido para o desenvolvimento satisfatório de um produto de software acarreta várias desvantagens, tais como:

- Dependência em pessoas específicas para a execução de tarefas pode ser um fator de risco para o normal andamento do projeto.
- Dificuldade para replicar uma determinada sequência de atividades.
- Impossibilidade de medição do desempenho do trabalho, dificultando o estabelecimento de relações de custo-benefício para uma posterior tomada de decisão.

- Uma vez que o processo não pode ser replicado nem medido em relação à eficácia e eficiência das atividades envolvidas, é praticamente inviável propiciar a sua melhoria, uma vez que não existe uma linha base concreta a partir da qual uma proposta de melhoria possa ser elaborada.

Em contrapartida, a definição e adoção de um processo adequado às características e **cultura** da organização, e às particularidades do projeto, possibilita o acesso à tecnologia e a construção de software cada vez mais complexo.

A cultura organizacional envolve artefatos (padrões de comportamento), valores éticos e morais compartilhados (crenças), políticas internas e externas e sistemas.
A cultura impõe regras que todos os membros dessa organização devem seguir e adotar como diretrizes e premissas para guiar seu trabalho.

Na busca por soluções às demandas, pesquisadores, desenvolvedores e demais profissionais se reuniram na Conferência sobre Engenharia de Software da OTAN (Organização do Tratado do Atlântico Norte), em 1968, onde o conceito e atribuições da Engenharia de Software foram oficialmente estabelecidos. A criação desta nova area surgiu em uma tentativa de contornar a crise do software e dar um tratamento de engenharia (mais sistemático e controlado) ao desenvolvimento de sistemas de software complexos.

Um sistema de software complexo se caracteriza por um conjunto de componentes de software abstratos (estruturas de dados e algoritmos) encapsulados na forma de procedimentos, funções, módulos, objetos ou agentes e interconectados entre si, que deverão ser executados em sistemas computacionais. O conjunto destes componentes e suas interconexiones determina a arquitetura do software.

Saiba mais

Enfrentando o Desafio: No Silver Bullet...

No referenciado artigo Não existe a bala de prata..., o autor analisa as diversas tecnologias apresentadas na época como supostas balas de prata que iriam a resolver problemas crónicos vinculados ao desenvolvimento de software. O autor argumenta que nenhuma ferramenta ou tecnologia irá produzir um aumento significativo (de ordens de magnitude) na produtividade do desenvolvimento de software uma vez que objetivam a denominada complexidade acidental, relacionada com as dificuldades encontradas na implementação. Por exemplo o tratamento para erros de compilação, poder de expressividade das linguagens de programação, etc. Neste contexto, Brooks discute porquê várias tecnologias não são a bala de prata.
- Ambientes de desenvolvimento integrado (IDEs);
- Linguagem de programação Ada;
- Orientação a Objetos;
- Inteligência artificial;
- Sistemas Especialistas;

Todas estas tecnologias enfocam na chamada complexidade acidental do software, e não nas dificuldades essenciais.
Brooks argumenta que as únicas ações que podem ter um impacto favorável significativo são aquelas que abordam um tratamento para lidar com a complexidade essencial, vinculadas a especificação, modelagem e testes.

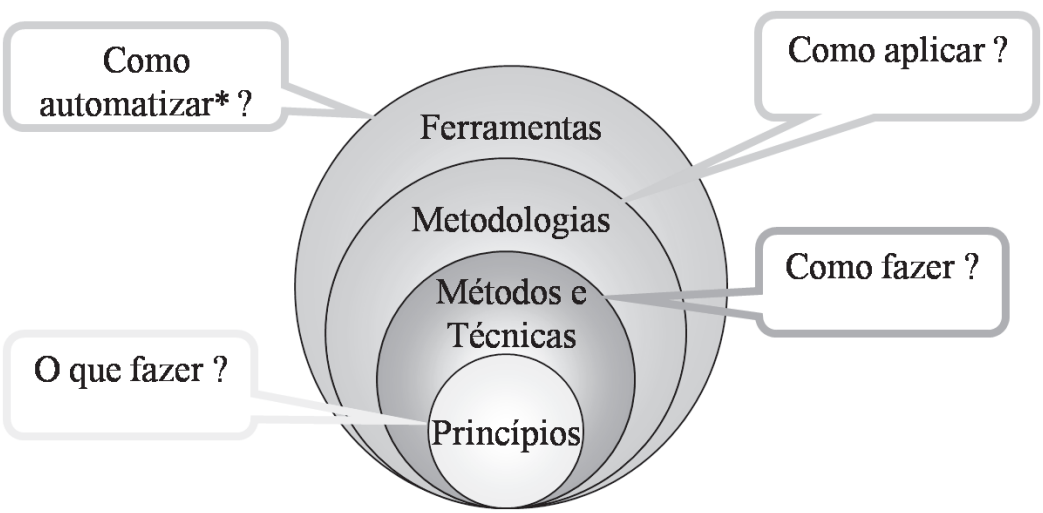
No Silver Bullet: Essence and Accidents of Software Engineering, by Frederick P. Brooks, http://www.lips.utexas.edu/ee382c-15005/Readings/Readings1/05-Broo87.pdf

Atividades de avaliação

1. Dê exemplos que ilustrem a influência do software na sociedade atual. Podem ser positivos ou negativos.
2. Leia o artigo de F. Brooks e faça uma análise crítica, fazendo uma analogia com soluções e ferramentas tecnológicas atuais tentando estabelecer em que medida respondem à problemática da complexidade dos sistemas atuais.
3. Dê exemplos de aspectos vinculados à complexidade essencial no desenvolvimento de sistemas.
4. Dê exemplos de técnicas e metodologias atuais que podem vir a resolver ou amenizar as dificuldades encontradas para lidar com a complexidade essencial no desenvolvimento de software.

2. Componentes da Engenharia de Software

A área de engenharia de software oferece insumos para o desenvolvimento de produtos de software, independentemente do domínio específico. Desta forma, um produto de software seja na área de redes, otimização ou inteligência artificial, precisa de tais insumos para atingir o patamar de qualidade esperado. Neste contexto, duas dimensões podem ser estabelecidas. Por um lado temos as atividades envolvidas no processo de desenvolvimento de um produto de software. Por outro temos as diferentes formas em que a Engenharia de Software dá apoio a essas atividades, como por exemplo através do estabelecimento de teorias e príncipios, definição de métodos e técnicas agregadas em metodologias, e ainda através do desenvolvimento de ferramentas. A ilustração a seguir representa os diversos aspectos do desenvolvimento de software abordados pela Engenharia de Software.



Princípios, de forma geral, consistem em práticas que têm se mostrado comprovadamente boas e de utilidade, e sua aplicação é recomendada para a realização de uma atividade. No contexto do desenvolvimento de software, algumas práticas são consideradas princípios a serem seguidos e aplicados no intuito de obter um produto de qualidade. Alguns desses princípios são brevemente definidos a seguir.

- **Modularidade –** Consiste na divisão de sistemas complexos em partes menores e mais simples com características desejáveis e bem definidas **(Coesão e Acoplamento)**. Decomposição é o ato de dividir um problema original em subproblemas, recursivamente. Composição é o ato de juntar os elementos componentes de um problema até chegar ao sistema completo. A modularidade tem como objetivo contribuir no entendimento de um problema complexo utilizando como estratégia a sua decomposição em partes menores. Adicionalmente contribui na manutenção e **reusabilidade** do sistema.
- **Separação de Conceitos –** Separar conceitos permite-nos trabalhar com aspectos individuais e diferentes de um mesmo problema. Esta separação facilita o entendimento, focando a atenção em certas características mais significativas, através de um processo de **abstração**.
- **Generalidade/Especialidade –** Soluções genéricas tendem a ser mais caras em termos de recursos e em tempo de desenvolvimento, ao contrário das soluções mais específicas. No processo de produção de software estas questões devem ser cuidadosamente analisadas.
- **Rigor e Formalidade –** O processo de software é uma atividade criativa tendendo naturalmente à imprecisão. O rigor é a abordagem que produz produtos mais confiáveis pelo controle das variáveis envolvidas. Formalidade implica que o processo esteja dirigido e avaliado por leis matemáticas.
- **Incrementabilidade –** Caracteriza o processo em modo passo-a-passo, incrementalmente e prevê que o objetivo desejado seja atingido por aproximações sucessivas. Esta abordagem pode ser muito útil quando os requisitos iniciais não foram todos obtidos antes do início do desenvolvimento da aplicação.
- **Antecipação de Mudanças –** Desenvolver o sistema visando possíveis evoluções do produto, de forma a tornar a sua manutenção e adaptabilidade mais fáceis e baratas. Quando o sistema é finalmente liberado, novos requisitos podem ser descobertos e requisitos antigos atualizados através do feedback do usuário.

Os princípios de Engenharia de Software parecem abstratos, mas tornam-se concretos quando aplicados no processo de produção do software. Por exemplo, na atividade de análise é possível utilizar o rigor, a abstração e a separação de conceitos na construção de modelos que tratam dos diferentes aspectos do sistema, como funções e fluxo de informações. Já no projeto é interessante o uso de princípios como rigor e formalidade na implementação e documentação dos modelos criados na análise, modularidade na construção do sistema e incrementabilidade nos testes.

Princípios são importantes para o sucesso no desenvolvimento de software, mas não são suficientes para obter produtos de qualidade. É preciso aliar Métodos, Técnicas e Ferramentas apropriadas para se aplicar os princípios no processo de produção de software.

Os Métodos da Engenharia de Software são abordagens estruturadas para o desenvolvimento de software, incluindo modelos, notações, etc.

Finalmente, na camada mais externa temos a família de ferramentas integradas que dão apoio às atividades relacionadas aos processos de engenharia são chamadas de CASE (Computer-Aided Software Engineering).

- Upper-CASE
- Ferramentas de suporte às atividades de processo nas fases iniciais de requisitos e projeto.
- Lower-CASE
- Ferramentas de suporte às atividades mais avançadas tais como programação, depuração e testes.

A utilização de ferramentas CASE é de fundamental importância para a correta aplicação dos métodos no contexto de metodologias específicas, visando o aumento na produtividade dos profissionais envolvidos.

Atividades de avaliação

1. Defina Engenharia de Software nas suas diversas dimensões, descrevendo as subáreas que a compõem e os relacionamentos entre elas.
2. Explique a importância da utilização dos princípios da engenharia para o desenvolvimento de aplicações nos diversos domínios.
3. Pesquise e defina os conceitos de coesão e acoplamento. Com base no entendimento destes conceitos, explique por que os princípios de baixo acoplamento e alta coesão são considerados boas práticas na implementação de sistemas computacionais.

4. Defina os princípios de abstração e reúso e analise os benefícios que podem ser alcançados pela sua utlização.
5. Cite algumas ferramentas genéricas (CASE) comumente utilizadas no desenvolvimento de sistemas, descrevendo as suas principais características e benefícios no contexto do desenvolvimento de software.

Síntese do capítulo

Nesta unidade é apresentada uma introdução à área de Engenharia de Software e sua contribuição no contexto da problemática do desenvolvimento de sistemas. Os conceitos básicos, tais como princípios, métodos e ferramentas são apresentados assim como também uma breve resenha histórica do seu surgimento como uma disciplina da Ciência da Computação.

Leituras, filmes e sites

Leitura

BROOKS Frederick P. No Silver Bullet: Essence and Accidents of Software Engineering. Computer, vol. 20, num. 4, pp. 10-19. April, 1986.

Referências

PRESSMAN, R.S. Engenharia de Software. 5ª ed. Rio de Janeiro: McGraw-Hill, 2002.

SOMMERVILLE, I. Engenharia de Software. 6ª ed. São Paulo: Addison Wesley, 2003.

PFLEEGER, S. Engenharia de Software. Teoria e prática. 2da edição. Pearson, 2004.

Capítulo 2

O Processo de Desenvolvimento de Software

Objetivos

- O desenvolvimento de software que culmina em um programa executável e satisfatório em termos de qualidade, envolve um conjunto de atividades cuja execução deve ser planejada de acordo às restrições impostas. O processo de desenvolvimento de software estabelece as atividades pertinentes ao desenvolvimento de um projeto de software, indicando quando e por quem precisam ser executadas de forma a atingir os objetivos de qualidade propostos.

1. Introdução

O desenvolvimento de software é uma atividade criativa e complexa cujo sucesso pode ser influenciado por diversos fatores que podem colocar em risco o atendimento aos objetivos propostos. Estatísticas indicam que um alto número de projetos não chegam ao fim, mais especificamente:

- Projetos que terminam dentro do prazo estimado: 10%
- Projetos descontinuados antes de chegarem ao fim: 25%
- Projetos acima do custo esperado: 60%
- Atraso médio dos projetos de 1 ano em relação ao prazo traçado originalmente.

The Chaos Report, 1984.

Entre as principais causas de estes problemas temos as seguintes:

- Metas e objetivos mal estabelecidos ou mal compreendidos pela equipe.
- Muitas atividades e pouco tempo.
- Informações insuficientes ou inadequadas.
- Estimativas intuitivas sem uso de dados históricos.
- Planejamento insuficiente.
- Produtos finais mal definidos.
- Equipe sem habilidades suficientes.
- Não são estabelecidos padrões de trabalho.

Um projeto ou empreendimento visa a criação de um produto ou a execução de um serviço específico, e temporário. Trata-se de um processo não repetitivo e que envolve um certo grau de incerteza na realização.

Diferentemente do desenvolvimento artesanal, a definição de um processo de desenvolvimento de software tem como objetivo a formalização das atividades relacionadas à elaboração de sistemas informáticos, visando responder a perguntas tais como: quando, como e por quem as atividades envolvidas no processo de desenvolvimento serão executadas. Adicionalmente, pontos de controle são estabelecidos de forma a determinar marcos a partir dos quais o andamento das atividades pode ser verificado.

A padronização e sistematização do processo de desenvolvimento de software trazem algumas vantagens, uma vez que é estabelecido o passo-a-passo a ser seguido para atingir o sucesso na execução de uma determinada atividade. A existência de uma definição do processo faz com que a execução das atividades independa dos profissionais envolvidos, propiciando um melhor compartilhamento e acessibilidade à informação, e sentimento de propriedade coletiva sobre o processo e o produto. Além disso, o processo pode ser reproduzido, ou replicado em diferentes cenários, medível em relação a desempenho, e, a partir da análise das medições, pode ser otimizado na busca pela melhoria contínua.

O processo de desenvolvimento de um produto de software geralmente é encarado como um projeto com inicio, médio e fim. Um projeto é caracterizado por uma sequência clara e lógica de eventos e atividades que precisam ser planejados e, durante sua execução, precisam ser controlados. Projetos de desenvolvimento de software surgem em resposta a uma necessidade ou uma oportunidade de negócio, e acontecem dentro de parâmetros predefinidos de tempo, custo, e escopo. O grau de atendimento a estes parâmetros representa um indicador da qualidade do processo adotado o que, consequentemente irá influenciar a qualidade do produto gerado como resultado e na satisfação do cliente.

A influência mais clara acontece em relação à definição do escopo, isto é, aos requisitos que precisam ser atendidos através das funcionalidades propiciadas pelo sistema. O atendimento aos requisitos é de fundamental importância para o cliente: se este aspecto não for satisfeito, o produto pode se mostrar inadequado às necessidades estabelecidas por ele como premissas do sistema.

De forma resumida, um processo de desenvolvimento envolve as diversas atividades, ao longo das quais, os custos da Engenharia de Software ficam aproximadamente distribuídos da seguinte forma:

- Especificação 10 %
- Projeto 30 %
- Desenvolvimento 20 %
- Testes 40 %

Estes custos podem ser influenciados pelos custos associados à qualidade, uma vez que, na medida em que tais atividades são desenvolvidas com o rigor e a profundidade adequados à sua complexidade, impactos positivos ou negativos podem ser acarretados nas atividades subsequentes. Por exemplo, se a atividade de especificação não for realizada de forma satisfatória, isto pode acarretar re-trabalho nas atividades de projeto e desenvolvimento. Com isso, uma aparente redução de custos na atividade de especificação pode implicar em um aumento significativo nos custos associados às outras atividades.

O fator humano

Em virtude do tamanho e complexidade, o desenvolvimento de sistemas de software é um empreendimento realizado em equipe. O sucesso de um projeto de software é baseado no relacionamento harmônico entre os componentes que podem fazer parte da equipe de desenvolvimento e dos clientes. São chamadas de stakeholders todas as pessoas que tem um interesse no desenvolvimento do projeto. Stakeholders podem ser classificados em dois grupos:

- Primários são aqueles que têm uma obrigação contratual ou legal com o projeto.
- Secundários são aqueles que têm apenas um interesse no projeto e não têm nenhuma relação formal.

Exemplos de stakeholders primários e secundários são listados a seguir:

Primários	Secundários
Gerentes e diretores seniores	Organizações sociais
Gerentes gerais	Organizações políticas
Gerentes funcionais	Ambientalistas
Gerentes de projeto	Concorrentes
Equipe de projeto	Comunidades locais
Cliente	Público em geral
Provedores, contratantes e subcontratantes	Grupos consumidores
Agências e comissões locais, estaduais e federais	Organizações profissionais
Organizações judiciais, legislativas e executivas	Institutos variados, como escolas e hospitais
Empregados	Mídia
Credores	Famílias
Acionistas	

Uma figura que se destaca pela sua função integradora e facilitadora para que o projeto seja concluído de forma satisfatória é o gerente.

As habilidades e a experiência necessárias para desempenhar o papel de gerente dependem das características específicas do projeto. Dependendo

destas características uma maior qualificação do gerente pode ser requerida. No entanto, dentre outras habilidades profissionais e inter-pessoais desejáveis em um gerente podem ser citadas:

- Mostrar capacidade de motivação e liderança.
- Facilidade de apresentação, comunicação e negociação.
- Facilidade para a solução de problemas.
- Influência na organização.
- Alerta às tendências sócio-econômicas.
- Ter experiência no domínio do aplicativo e no desenvolvimento de software
- Ter habilidades de análise e gerenciamento de riscos, estimativa, planejamento e análise de decisões
- Ter boa capacidade de gerenciamento de tempo e custo.
- Ser pragmático no escopo e na implementação de planos e completamente honesto e objetivo na avaliação dos resultados e avaliação do trabalho.

O papel gerente tem as mais variadas responsabilidades em relação ao planejamento, monitoramento e controle das atividades de projeto. Dentre outras funções temos: alocação de recursos, estabelecimento de prioridades, coordenação das interações com clientes e usuários, membros da equipe e alta direção, delineamento das metas que orientam a equipe do projeto, etc. O gerente de projeto também estabelece o conjunto de práticas que garantem a integridade e a qualidade dos artefatos do projeto.

Responsabilidade profissional e ética do engenheiro de software

Considerando o papel central que computadores desempenham na sociedade atualmente, os engenheiros de software representam peças-chave, uma vez que contribuem e participam ativamente de todo o processo de desenvolvimento de sistemas.

Consequentemente surge o compromisso de fazer da Engenharia de Software uma profissão benéfica e respeitada. Esse compromisso foi formalizado através do enunciado do Código de Ética e da Prática Profissional, apresentado resumidamente a seguir.

Saiba mais

Código de Ética e de Prática Profissional

PÚBLICO. Agir consistentemente com o interesse público.
CLIENTE E EMPREGADOR. Agir em conformidade com os melhores interesses de cliente e empregador em consistência com o interesse público.
PRODUTO. Assegurar que seus produtos cumpram o mais alto padrão profissional possível.
JULGAMENTO. Manter integridade e independência em seu julgamento profissional.
GERENCIAMENTO. Promover abordagem ética para o gerenciamento do desenvolvimento e manutenção do software.
PROFISSÃO. Fomentar a integridade e reputação da profissão, de modo consistente com o interesse publico.
COLEGAS. Ser justo e dar apoio aos colegas.
PESSOAL. Participar de aprendizado constante com relação à pratica da sua profissão.

Software engineering code of ethics is approved. Communications of the ACM, vol. 42, issue 10 , pp. 102 – 107, 1999.

Atividades de avaliação

1. Escolha um dos aspectos citados no Código de Ética do Engenheiro de Software e analise a sua importância. Exemplifique atitudes que podem ser classificadas como éticas e não-éticas em relação ao aspecto do Código escolhido. Justifique.
2. Para os empreendimentos a seguir determine os stakeholders, classificando-os em primários e secundários.
   a) Projeto de sistema de controle hospitalar.
   b) Projeto de sistema de controle acadêmico.
   c) Projeto de desenvolvimento de uma nova linha de carros.
   d) Projeto de implantação de uma fábrica de reciclagem.
3. Você é um engenheiro de software envolvido no desenvolvimento de um sistema de controle de manufatura de peças de carros. Durante a instalação você descobre que a operação do sistema tornará dispensável o trabalho de muitos funcionários. Esta suspeita entre os funcionários acaba dificultando o seu acesso a informações importantes. Qual seria a sua postura, como engenheiro, nesta problemática?

Diferentemente de projeto, o conceito de processo ou operação envolve o trabalho realizado de modo contínuo, resultando na produção de um mesmo produto ou prestação de um serviço determinado, sem um prazo definido para término. Por exemplo, a construção de carros em uma linha de montagem é um processo, já o desenvolvimento de um novo modelo de carro é um projeto.

2. Fases de desenvolvimento de software

De forma geral, todo desenvolvimento de software acontece no contexto de um projeto que objetiva a criação de um produto original ou a realização de um serviço. Um projeto é caracterizado por ser temporário e não repetitivo, e que pressupõe certo grau de incerteza na sua realização.

As tarefas envolvidas no projeto normalmente são executadas por pessoas, e sua execução é restringida no prazo, custo e escopo. Como em qualquer empreendimento, as atividades precisam ser planejadas, implementadas e, controladas durante sua execução. Os resultados gerados a partir da execução de uma tarefa geralmente se constituem em insumos para outras tarefas.

O projeto para o desenvolvimento de um produto de software pode surgir a partir de dois estímulos:

- Necessidade proveniente de uma entidade externa ou cliente,
- Oportunidade de negócio interna à organização ou grupo que empreenderá o projeto.

A partir da identificação da necessidade ou da oportunidade, e antes mesmo do projeto de desenvolvimento dar inicio, a viabilidade do empreendimento precisa ser determinada. A conclusão satisfatória a partir do relatório gerado como resultado estabelece o comprometimento da organização em dar continuidade ao projeto. O estudo da viabilidade de um projeto pretende responder a questões tais como:

- O sistema contribui para os objetivos gerais da organização?
- O sistema pode ser implementado dentro das restrições de tempo, custo e levando em conta a tecnologia disponível?
- O sistema pode ser integrado a outros sistemas existentes na organização?

A partir das informações coletadas é formulado um relatório que será utilizado como base para a tomada de decisão em relação à continuidade do projeto ou não.

A coleta de informações que podem vir a auxiliar na resposta às questões acima pode ser realizada a partir da elaboração de questionários aplicados a gerentes e demais interessados que podem se tornar futuros usuários finais do sistema, engenheiros de software e demais especialistas em tecnologia. O relatório ainda pode sugerir alterações de escopo, prazo e custo de forma a melhor se adequar à realidade da organização.

É comum fazer-se aqui uma estimativa dos esforços a serem despendidos, especialmente em termos de recursos necessários e estimativas de custo e prazo, para dar subsídios à autorização do projeto.

Com base nas informações coletadas, a viabilidade do projeto é avaliada de forma a determinar o custo-benefício que a execução do projeto vai acarretar para as partes envolvidas. A partir desta análise, e em caso positivo, é assinado o **termo de abertura** formalizando a aprovação do projeto e autorizando o inicio das atividades.

O Termo de Abertura é um documento ligado à formalização da existência do projeto dentro da organização, autorizando seu inicio. Descreve as necessidades do negócio que será atendida, identificando restrições e premissas impostas ao projeto. Através deste documento é designado um gerente para o projeto, e sua autoridade e atribuições são especificadas.

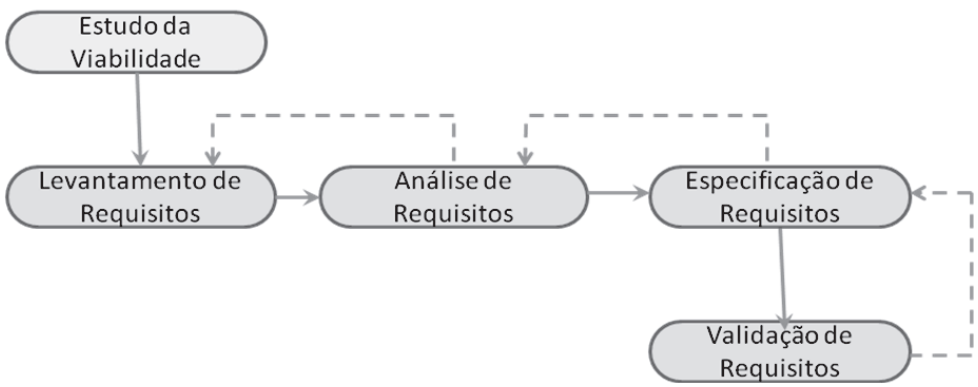
Atividades de avaliação

1. O que é o estudo da viabilidade e para que serve?
2. Para cada cenário a seguir indicar si se trata de um processo ou um projeto. Justifique.
   a) Ampliar a casa.
   b) Pagamento do condomínio do apartamento.
   c) Levar as crianças ao cinema.
   d) Pegar as crianças da escola.
   e) Passear o cachorro todo dia.
   f) Levar o carro a oficina para revisão de rotina.
   g) Furar pneu.
   h) Escrever um livro.

Uma vez iniciado o projeto, as atividades podem ser agrupadas em 3 macro-fases, são elas: definição, desenvolvimento e manutenção. Cada uma destas macro-fases envolve a execução de tarefas específicas.

2.1. A fase de definição

A fase de definição objetiva estabelecer da forma mais clara possível o que será desenvolvido. Neste primeiro momento é preciso determinar a informação a ser processada, função e desempenho desejados, comportamento esperado do sistema, interfaces a serem estabelecidas, restrições de projeto e critérios de validação exigidos.



A disciplina de Gestão de **Requisitos** objetiva manter sob controle o conjunto de requisitos no intuito de reduzir a sua instabilidade de forma a manter os custos dentro do orçamento previsto.

Como resultado da fase de definição espera-se obter uma especificação completa do software a ser desenvolvido estabelecendo de forma clara e não-ambígua quais serviços são requeridos e as restrições a serem consideradas sobre o desenvolvimento e operação do sistema de software. A fase de definição é baseada nos processos de Engenharia de **Requisitos**, são eles.

- Levantamento e análise de requisitos
- Especificação de requisitos
- Validação de requisitos

Os processos de engenharia de requisitos são particularmente críticos, uma vez que qualquer erro ou omissão leva inevitavelmente a problemas nas fases posteriores de desenvolvimento.

Saiba mais

Requisitos funcionais vs. Requisitos não funcionais

Requisitos de software são descrições e especificações de um sistema descrevendo suas características ou funcionalidades necessárias para atingir seus objetivos. Adicionalmente, restrições sobre o sistema geradas durante o processo de engenharia de requisitos podem ser incluídas. Os requisitos podem ser classificados em dois tipos: requisitos funcionais e não funcionais.

Os requisitos funcionais descrevem funcionalidades ou serviços que o sistema deve fornecer, como o sistema irá a reagir a entradas particulares e como irá a se comportar em determinadas situações. Dependem do tipo de software, dos usuários e tipo de sistema onde o software é usado.

Exemplos de Requisitos Funcionais:

- O usuário deve ser capaz de efetuar consultas sobre todo o conjunto inicial de informações de dados ou selecionar um subconjunto deste.
- O sistema deve fornecer visualizadores apropriados para o usuário ler documentos no arquivo de documentos.
- Cada pedido deve possuir um identificador único que o usuário deve ser capaz de copiar para a área de armazenamento permanente de contas.

Os requisitos não-funcionais podem ser definidos como restrições sobre os serviços ou funções oferecidos pelo sistema tais como restrições temporais, sobre o processo de desenvolvimento, padrões ou normas a serem seguidos, etc. Os requisitos não-funcionais podem ser classificados em:

- Requisitos de Produto: Requisitos que especificam como o produto deve se comportar; e.g. velocidade de execução, tempo de resposta, confiabilidade, navegabilidade, etc.
- Requisitos Organizacionais: Requisitos que são consequência de políticas organizacionais, e.g., padrões de processo usados, requisitos de implementação, etc.
- Requisitos Externos: Requisitos levantados a partir de fatores externos ao sistema e ao processo de desenvolvimento, e.g., requisitos de interoperabilidade, requisitos legislativos, etc.

Os requisitos, tanto funcionais como não-funcionais, podem precisar ser totalmente satisfeitos, ou considerados altamente desejáveis, podendo ser eliminados.

O levantamento de requisitos é a atividade que objetiva a compreensão do problema aplicada ao desenvolvimento de software. As necessidades ou requisitos do sistema são levantados e definidos a partir do domínio do negócio. O principal objetivo deste processo é que usuários e desenvolvedores tenham a mesma visão do problema a ser resolvido, sem levar em conta o ambiente tecnológico, inicialmente.

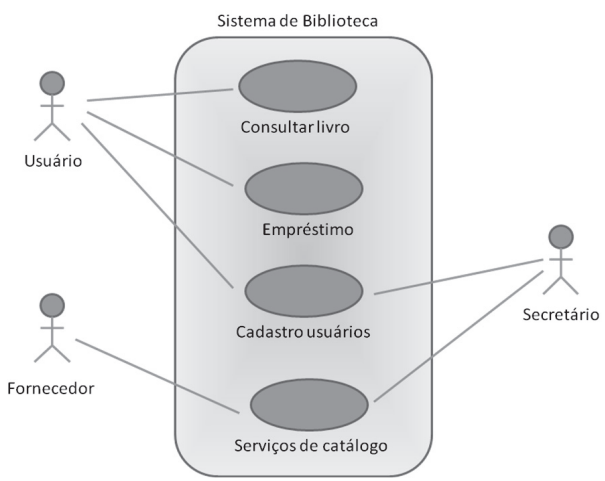
O documento de **especificação de requisitos** detalha o resultado da análise e especificação de requisitos e das outras atividades da fase de definição, uma vez validado pelos clientes, representa a concordância entre clientes, analistas e desenvolvedores sobre o que deve ser desenvolvido.

O fluxo de levantamento de requisitos objetiva obter o enunciado completo, claro e preciso dos requisitos de um produto de software. O resultado principal do fluxo é o documento de **Especificação de Requisitos de software.**

A atividade de levantamento de requisitos é considerada muito difícil e crítica para o sucesso do projeto, uma vez que estabelece os pilares sobre o quais o sistema vai ser construído. Adicionalmente, existe uma dificuldade intrínseca na captura dos requisitos devido ao fato dela acontecer em uma etapa muito inicial do processo, onde ainda existe pouca informação disponível. Adicionalmente, o fato de se tratar de uma atividade fortemente baseada na comunicação, faz com que seja muito propensa a erros. Existem diversas técnicas para auxiliar no levantamento de requisitos. Dentre as técnicas mais comumente utilizadas temos a aplicação de questionários e realização de entrevistas a clientes e usuários finais. Adicionalmente a demonstração de tarefa por parte de usuários e a observação por parte dos analistas pode ser de utilidade para compreender o fluxo de atividades a ser automatizado pelo sistema. Outras técnicas mais elaboradas tais como substituir o usuário (role playing), brainstorming e brainwriting, prototipação, workshops ou oficinas de requisitos podem ser aplicadas para um maior aprimoramento dos requisitos levantados. Um estudo em empresas e produtos similares pode ser útil no caso do desenvolvimento de aplicativos de prateleira, onde não existe um cliente específico destinatário da aplicação.

Modelos são representações simplificadas da realidade com a intenção de facilitar o processo de entendimento de forma a lidar melhor com a complexidade.
De forma geral, vários modelos complementares precisam ser gerados para modelar um sistema complexo. Modelos são gerados como resultado de um processo de abstração.

Técnicas de modelagem são utilizadas para auxiliar, não somente no processo de definição dos requisitos de usuário e especificação dos requisitos do sistema, mas servem de entrada para diversas atividades ao longo do processo de desenvolvimento de software.



A linguagem Unified Modeling Language (UML) é uma linguagem de modelagem amplamente utilizada para modelar e documentar as diferentes fases do desenvolvimento de software. Em particular, foi estabelecida pela Object Management Group (OMG) como padrão para a modelagem de sistemas orientados a objetos em 1997.

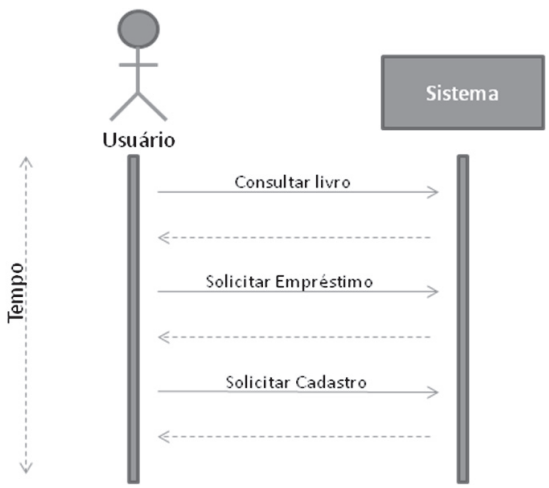
Dentre os modelos normalmente gerados nas fases iniciais de levantamento de requisitos temos o dia-

grama de casos de uso. O diagrama de casos de uso fornece uma visão do comportamento visível do sistema, descrevendo as suas interações com o mundo exterior. As principais funcionalidades do sistema e os atores envolvidos são representados interagindo em casos de uso. Segue um exemplo de diagrama de caso de uso para um sistema de biblioteca.

Normalmente, os casos de usos são acompanhados por descrições textuais. A UML não fornece um modelo específico para estas descrições, que podem ser redigidas em linguagem natural ou fazendo uso de uma linguagem mais estruturada.

A fase de análise de requisitos envolve o estudo detalhado dos requisitos levantados e a construção de novos e mais refinados modelos representando os conceitos ou entidades relevantes do domínio do problema, os quais são candidatos a entidades no domínio da solução computacional (sistema).

A modelagem gráfica é muito útil para o detalhamento de sequências de ações acontecendo ao longo do tempo. Neste contexto, o diagrama de sequência ilustrado ao lado, apresenta uma interação na forma de uma sequência temporal de mensagens sendo enviadas entre objetos instância durante um cenário de execução.



Diagramas de sequência são utilizados para detalhar casos de uso, possibilitando uma análise mais aprofundada do comportamento do sistema.

Uma vez especificados, os requisitos precisam ser validados junto aos clientes e usuários de forma a verificar a sua consistência e aderência às expectativas.

No caso do desenvolvimento orientado a objetos, as entidades são mapeadas para classes com atributos e métodos, as quais irão modelar os objetos gerados como instâncias. As associações entre classes determinam a forma em que os objetos irão colaborar, em tempo de execução, para a realização dos casos de uso.

Os modelos gerados nesta fase constituem um refinamento dos modelos gerados na fase de levantamento de requisitos na medida em que são mais precisos e detalhados. Os modelos precisam ser verificados e validados em relação à especificação dos requisitos obtida no fluxo de levantamento de requisitos. O modelo de análise gerado como resultado objetiva a modelagem conceitual do conhecimento do domínio do problema em um conjunto de abstrações e seus relacionamentos, e a coleta de informações para a definição dos seus atributos.

Atividades de avaliação

1. Quais são os processos que fazem parte na Engenharia de Requisitos? Conceitue.
2. Dado um sistema de controle de estoque de uma mercearia estabeleça:
   a) Stakeholders ou interessados primários e secundários.
   b) Requisitos funcionais.
   c) Requisitos não funcionais.
3. Para um sistema de venta de livros online escreva um conjunto de requisitos não funcionais, estabelecendo a confiabilidade esperada e tempo de resposta.
   a) Desenhe o diagrama de casos de uso.
   b) Desenhe o diagrama de sequência de um caso de uso da sua escolha.
4. A partir da sua experiência no uso de um caixa eletrônico, desenhe um diagrama de casos de uso de forma a auxiliar no entendimento dos requisitos do sistema.
5. Faça uma pesquisa sobre uma das técnicas citadas para auxiliar no levantamento de requisitos.
6. Forneça um exemplo de sistema no qual fatores sociais e políticos podem influenciar fortemente nos seus requisitos.
7. Usando seu conhecimento sobre o sistema acadêmico da sua universidade, desenvolva um conjunto de casos de uso como base para o entendimento dos requisitos envolvidos.
8. Considere o sistema anterior interagindo com um sistema de controle financeiro relativo ao pagamento de matrículas. Como este novo aspecto se reflete no diagrama de casos de uso.

2.2. A fase de Desenvolvimento

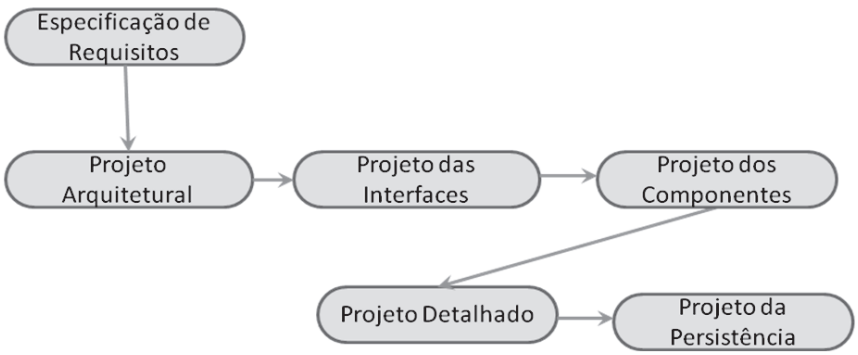
A fase central de desenvolvimento do produto focaliza em como o software será construído, envolvendo as atividades de Projeto, Implementação, Testes e Implantação.

Durante o Projeto determina-se como o sistema funcionará para atender à especificação, considerando os recursos tecnológicos existentes. A fase de projeto tem por objetivo definir uma estrutura implementável computacionalmente para um produto de software que atenda aos requisitos especificados na fase de análise. Esta fase tem seu foco na concepção de estruturas e

Os padrões de projeto são soluções abstratas e reutilizáveis dirigidas a um problema de projeto recorrente.
Através dos padrões de projeto são identificadas e especificadas as abstrações que fazem parte da solução em um nível superior ao de classes, instâncias ou componentes.

mecanismos robustos que tornem a implementação mais confiável e produtiva. Nesta fase são consideradas as características da plataforma que será utilizada e as restrições decorrentes que podem influenciar no desempenho e demais aspectos não funcionais do sistema sendo desenvolvido.

As atividades principais envolvidas incluem o Projeto da **arquitetura** e o Projeto detalhado.

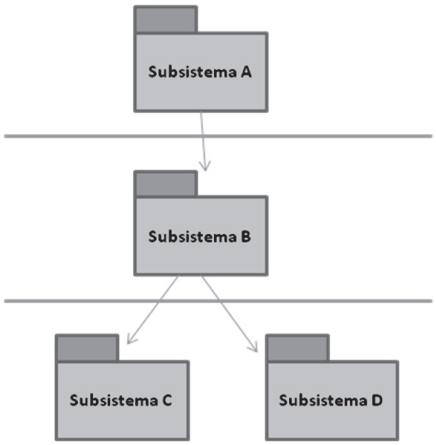


No projeto arquitetônico são abordados os aspectos estratégicos de projeto externo e interno, definindo a divisão do produto em subsistemas e escolhendo as tecnologias mais adequadas. Esta divisão visa facilitar o trabalho de implementação uma vez que promove o reuso. A prática de reuso é sempre recomendada uma vez que garante que os componentes sendo reutilizados foram testados e avaliados de forma satisfatória em diferentes contextos, aumentando, consequentemente, a qualidade do produto sendo construído. A prática de reuso pode ser aplicada nos diferentes níveis de abstração e fases. Por exemplo, o projeto pode ser orientado ao reaproveitamento de abordagens e **soluções de colaborações entre classes para resolução de problemas de desenho** típicos visando adicionalmente à reutilização de código (classes, objetos e plataformas).

Uma estruturação típica em camadas segue a seguinte organização:

- Camadas de aplicação envolvendo os componentes que implementam as interfaces do sistema com o exterior (Camada de fronteira) e os que implementam a lógica da aplicação propriamente dita (Camada de controle).
- Camadas de domínio envolvendo os componentes que implementam o acesso à informação fisicamente armazenada (Camada de entidade).
- Camadas da implementação envolvendo os componentes de informação armazenados fisicamente na memória (Camada de persistência) e as bibliotecas e demais utilitários (Camada de sistema).

Utilizando a linguagem de modelagem UML, a seguinte notação é utilizada para reprentar pacotes e subsistemas.



Uma vez que a definição dos componentes é concluída, estes precisam ter suas interfaces bem definidas de forma a possibilitar a correta troca de informações internamente entre os componentes do sistema. Analogamente, o projeto das interfaces do sistema com o exterior é trabalhado nesta fase e envolve estudos específicos sobre usabilidade, facilidade de aprendizado, produtividade dos usuários em relação ao desempenho e tempo de resposta do sistema. Dependendo do tipo de aplicação sendo desenvolvida, o projeto da interface de usuário pode demandar mais aprofundamento, por exemplo, no caso de uma aplicação web se comparado a uma aplicação desktop.

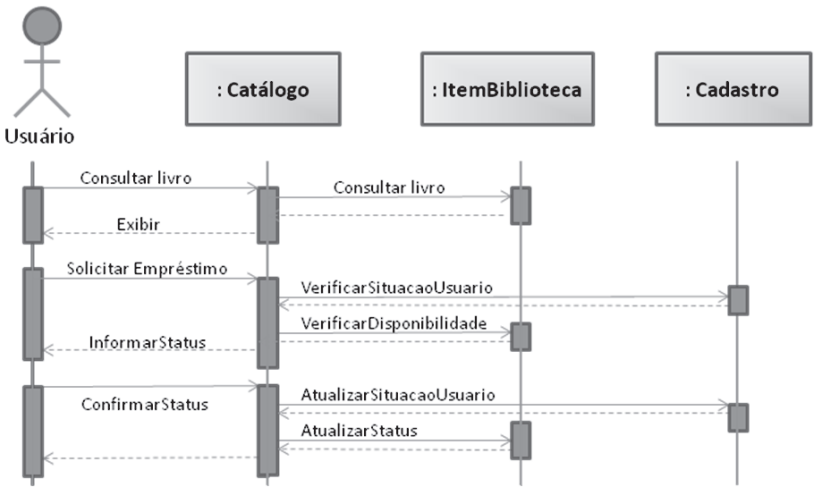
O projeto detalhado envolve uma análise detalhada dos artefatos gerados nas atividades anteriores, de forma a construir o conhecimento necessário para a implementação do sistema. Em particular, o detalhamento dos casos de uso visa à resolução de detalhes dos fluxos envolvidos, considerando os componentes, suas interfaces e os relacionamentos entre eles.

Um módulo é considerado um componente do sistema que fornece um ou mais serviços para outros módulos. A partir desta decomposição são determinadas as entidades de software necessárias para a realização dos casos de uso. A informação contida no projeto detalhado é de fundamental importância para projetistas e desenvolvedores.

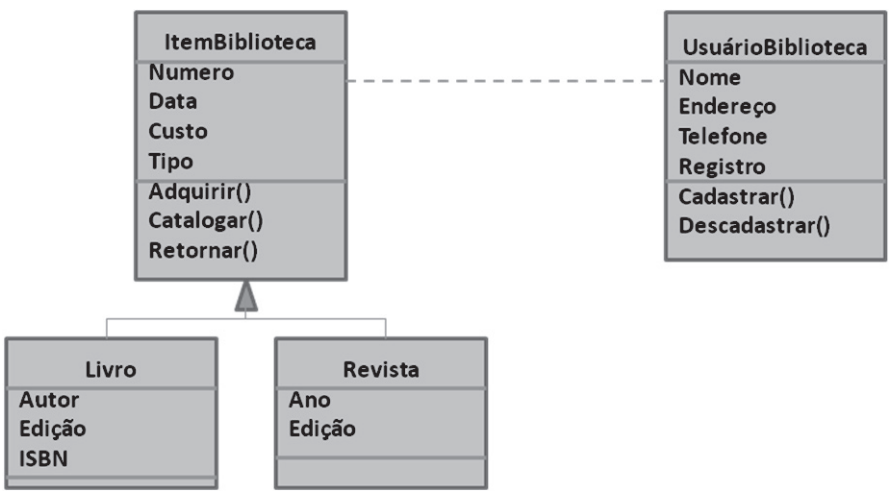
No contexto do paradigma orientado a objetos, o projeto das entidades envolve o mapeamento das classes conceituais para classes de projeto, cujos objetos instância se comunicam através de interfaces bem definidas.

As vantagens da abordagem orientada a objetos são bem conhecidas. Comumente, objetos são representações do mundo real, portanto, a estrutura do sistema pode ser rapidamente modelada e entendida. Por outro lado, o

encapsulamento propiciado pela abordagem possibilita que mudanças na implementação dos objetos possam ser feitas sem impactar em outros objetos, sempre que a interface não for afetada. Segue o diagrama de sequência para os casos de uso consultar e solicitar empréstimo no sistema biblioteca.



No mundo orientado a objetos, esta informação é organizada em um modelo de classes refinado e sua descrição textual. Este modelo ilustra os conceitos significativos em um domínio de problema representando as coisas do mundo real e sua transição para o universo computacional, e não de componentes de software. Segue o modelo de objeto parcial em UML, do sistema de biblioteca.



Uma decomposição orientada a objetos na fase de projeto incorpora classes de objetos, seus atributos e operações. Neste exemplo, a classe ItemBiblioteca tem, além dos atributos, várias operações associadas que implementam as funcionalidades do sistema.

Finalmente, o Projeto da persistência envolve o projeto das estruturas externas utilizadas para o armazenamento persistente das informações. A maioria dos sistemas de software utiliza bancos de dados para o armazenamento das informações. A definição lógica dos dados processados pelo sistema é chamada de modelo semântico dos dados. A técnica de modelagem de dados mais amplamente utilizada é a **modelagem Entidade-Relacionamento-Atributo (ERA).**

A modelagem ERA foi proposta a meados da década de 1970 por Chen, sendo que a posterior variação proposta por Codd em 1979 foi muito popularizada.
Os modelos ERA continuam a ser amplamente utilizados no projeto de banco de dados

Embora a linguagem UML não incorpora uma notação específica para a modelagem de banco de dados, uma variação do diagrama de classes pode ser utilizada para a modelagem. Neste caso, as entidades são representadas como classes sem operações, onde os atributos são representados como atributos da classe, no compartimento intermediario, mas sem operações. Adicionalmente, as associações identificadas entre as classes representam as relações.

Finalmente é feita a revisão do projeto, validando os resultados do projeto, confrontando-os com os resultados dos Requisitos e da Análise.

É importante salientar que todas as decisões de projeto precisam ser explicitamente documentadas, de forma a disponibilizar o conhecimento e o aprendizado adquirido na tomada de decisão. Desta forma o conhecimento adquirido através da experiência pode ser reutilizado em uma situação de projeto similar, e, de forma geral, contribuindo para o conhecimento da equipe que participa do projeto.

Atividades de avaliação

1. Descreva quais são as principais atividades envolvidas na fase de projeto.
2. Dado o sistema de controle de estoque especificado em exercícios anteriores, identifique os principais componentes envolvidos no projeto arquitetural global do sistema.
3. Idem para o sistema de controle acadêmico.
4. No contexto da orientação a objetos, determine as principais entidades (classes de objetos) que participam dos sistemas de controle de estoque e controle acadêmico, cujas arquiteturas foram modeladas nos itens anteriores.

Uma vez que os artefatos produzidos como resultado das atividades de projeto atingem o nível de refinamento requerido, a fase de implementação pode ser iniciada. Esta fase tem como objetivo a tradução da descrição computacional gerada na fase anterior em código executável. Esta fase depende fundamentalmente da plataforma de desenvolvimento escolhida e suas características em relação à escolha por ferramentas e bibliotecas disponíveis.

Em esta fase, a utilização e boas práticas de programação podem ser de grande utilidade para o aumento da produtividade dos desenvolvedores. Exemplos de boas práticas de programação incluem: manter o código simples, uso de padrões de indentação, nomes de variáveis e demais estruturas que reflitam seu conteúdo, etc. Outras práticas como a programação em pares e uso de **refatoração** popularizadas pelas **metodologias ágeis como XP** podem trazer benefícios adicionais.

A refatoração (do inglês refactoring) consiste em pequenas modificações no código de forma a melhorar a estrutura interna do programa sem alterar seu comportamento externo. O objetivo primário da refatoração é tornar o código mais fácil de entender.

Ferramentas CASE para a geração de código a partir de modelos podem ser utilizadas de forma a agilizar o processo de implementação.

Exemplos deste tipo de ferramentas recebem como entrada os artefatos de modelagem ou diagramas em UML e geram código executável em linguagens como java ou C. De forma geral, a extensão do código gerado depende da qualidade e detalhamento dos diagramas utilizados como entrada para o processo.

O desenvolvimento dirigido por modelos ou MDD é baseado no princípio de considerar artefatos de modelagem como entidades de primeira classe. É a partir destas entidades que, através de transformações entre modelos, é gerado o código em linguagem de programação.

Os métodos ágeis tem seu foco no desenvolvimento e entregas rápidas de software ao cliente. Baseado no método iterativo e incremental, a metodologia ágil. Alguns dos princípios dos métodos ágeis são: envolvimento do cliente, entrega incremental, valorização das pessoas sobre os processos, aceite a mudanças e manter a simplicidade do código. eXtreme Programing ou XP é o representante mais popular das metodologias ágeis.

Dependendo do modelo ou ciclo de vida adotado, a construção do sistema pode ser realizada ao longo de entregas parciais, onde cada uma delas é encarada como um subproduto que precisa ser testado e integrado até que a implementação da totalidade do sistema seja atingida.

Uma vez que a construção do sistema é concluída é preciso realizar a sua **verificação e validação** (V & V) de forma a garantir que o sistema funciona de forma adequada em relação à ocorrência de falhas ou defeitos, e que realiza as funcionalidades previstas na sua especificação. Ambos os aspectos são de fundamental importância para a satisfação do cliente e crítico para o desenvolvimento de software. A partir da execução dos processos de V & V pode ser estabelecido um nível de confiabilidade no produto desenvolvido em relação à sua adequação para o uso pretendido. De forma geral, o nível de confiabilidade depende da natureza do sistema desenvolvido.

Durante os processos de verificação e validação é determinado se o sistema funciona conforme com a sua especificação e satisfaz os requisitos do cliente. Estes processos envolvem a realização de atividades de revisão e a execução do sistema contra casos de teste derivados da especificação.

Validação: estamos construindo o produto correto?

Verificação: estamos construindo o produto corretamente?

A V & V começa com revisões nos requisitos e continua ao longo de revisões de projeto e inspeções de código até o teste do produto. Na medida em que os componentes são desenvolvidos, inspeções de código e testes de unidade na procura por defeitos precisam ser realizados de forma a garantir a corretude do código. Posteriormente, os componentes verificados são inte-

grados em componentes maiores, envolvendo a execução de novos processos de revisão e verificação.

Inspeções ou revisões de software são técnicas de V & V estáticas, uma vez que o sistema não precisa ser executado no computador. Estas técnicas consistem na análise e verificação dos artefatos gerados nos diversos estágios de desenvolvimento, tais como especificações, diagramas e código fonte, com o objetivo de encontrar erros, omissões e anomalias. Adicionalmente, as inspeções podem contribuir em aspectos mais amplos de qualidade de um programa como aderência a padrões, eficiência dos algoritmos, manutenibilidade e considerações sobre estilo de programação.

A realização de inspeções de software é complementada pela realização de testes no sistema implementado. A atividade de testes consiste em executar o sistema, ou parte do sistema, com dados de teste fornecidos como entrada (técnica dinâmica). As saídas resultantes são analisadas no intuito de determinar se o comportamento e desempenho do sistema estão de acordo ao esperado.

Atividades de avaliação

1. Explique as diferenças entre verificação e validação.
2. Caracterize as atividades de inspeção e de testes. Explique por que a inspeção é considerada uma técnica de V & V estática, enquanto que a de testes é dinâmica.
3. Considerando que a técnica de inspeção de software é estática, estabeleça quais tipos de erros ou defeitos dificilmente podem ser detectados através de inspeções.
4. Analisadores estáticos são ferramentas CASE utilizadas para automatizar o processo de verificação de um programa. Explique o princípio de funcionamento deste tipo de ferramenta.

Considerando que os testes não podem ser executados de forma exaustiva uma vez que demandaria muito trabalho e, consequentemente, um investimento que dificilmente um cliente estaria disposto a alavancar, os testes devem ser planejados cuidadosamente. A atividade de planejamento dos testes objetiva que, com a elaboração e execução do menor número de testes, seja possível verificar a maior parte do código de um sistema e detectar a maior quantidade de defeitos. Esta propriedade é chamada de **cobertura** dos testes, que deve ser maximizada.

Testes devem ser planejados e projetados objetivando detectar o maior número de defeitos, uma vez que a comprovação de que o sistema não apre-

senta defeitos é impossível de ser demonstrada. Um teste pode ser avaliado como satisfatório quando possibilita a detecção de erros no sistema.

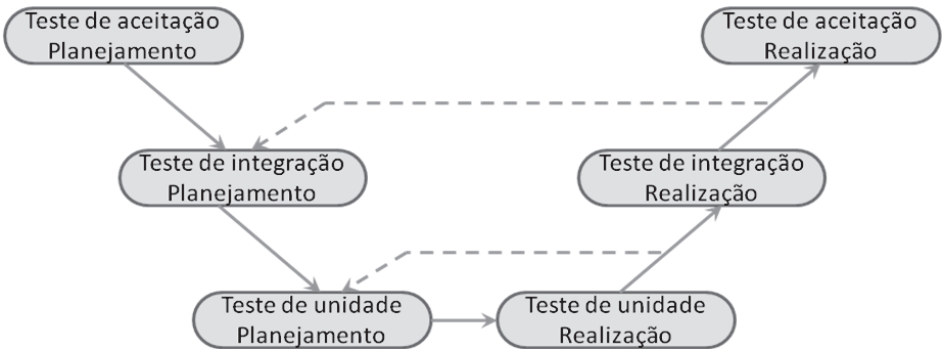
Considerando que os testes precisam ser automatizados para alcançar o máximo beneficio durante a sua execução e de forma que possam ser replicados, estes precisam ser implementados. O desenvolvimento dos testes segue o mesmo processo de desenvolvimento que qualquer outro tipo de software abordando as atividades de planejamento, projeto, implementação, execução e verificação.

O grau de rigor aplicado no desenvolvimento das atividades de teste depende do tipo de aplicação. Aplicações cuja importância pode ser crítica, com fortes restrições de tempo e precisão das quais podem depender, por exemplo, a integridade de vidas humanas, requerem de baterias de testes criteriosas e extensas de forma a garantir que a qualidade do sistema é satisfatória. Exemplo de este tipo de aplicações pode ser: sistemas controladores da aeronáutica, controladores de dispositivos hospitalares, sistemas de monitoramento de segurança, etc.

De forma geral, o desenvolvimento de qualquer sistema envolve a execução dos denominados testes de unidade (ou módulos), onde o desenvolvedor testa a unidade de código desenvolvida na procura por defeitos e bugs. Na medida em que o desenvolvimento avança, novas unidades são desenvolvidas e testadas, e integradas ao sistema. Toda vez que uma integração acontece, embora ou unidade tenha sido devidamente testada, o sistema precisa ser novamente avaliado de forma a garantir que nenhum defeito foi introduzido. Estes são denominados de testes de integração. Os testes de unidade e de integração são classificados como testes de caixa de branca, uma vez que para a correta execução dos mesmos por parte da equipe de desenvolvimento é preciso ter acesso direto ao código fonte sendo testado.

Depois de o sistema ser testado na integridade pela equipe de desenvolvimento, o cliente precisa avaliar se o produto que está sendo entregue se encontra de acordo com o que foi solicitado, levando em conta a especificação realizada durante a fase de levantamento de requisitos. O teste executado pelo cliente no ambiente de trabalho é chamado de teste de aceitação ou teste alfa. Estes testes são executados pelos usuários finais do sistema, os quais irão executar e verificar que todas as funcionalidades foram implementadas de forma satisfatória de acordo com a expectativa do cliente.

Testes de aceitação são classificados como testes de caixa preta, uma vez que não é necessário conhecer ou acessar o código para que possam ser realizados. Diferentemente, o sistema é testado através da interface disponível, a partir da qual o usuário/cliente do sistema pode alimentar o sistema com informações de entrada, as quais serão processadas gerando as saídas.



O gráfico em V dos testes representa a ordem em que as atividades de planejamento e realização dos respectivos testes acontece na prática. A partir da figura podemos perceber que a ordem em que as atividades de planejamento das diferentes baterias de testes acontecem, é inversa em relação à ordem em que as baterias são realizadas. Isto é, inicialmente é feito o planejamento dos testes de aceitação que serão realizados na hora da entrega do sistema ao cliente, no entanto estes testes são os últimos a serem executados. O planejamento dos casos de teste normalmente é baseado nos cenários específicos modelados pelos casos de uso.

Os testes executados antes da liberação de uma nova versão do sistema têm como meta aumentar a confiança do fornecedor de que o sistema atende aos requisitos. Neste sentido, os testes executados devem objetivar a busca por falhas ou defeitos no sistema, alimentados com entradas com alta probabilidade de gerar falhas ou saídas não previstas. Algumas diretrizes para testes bem-sucedidos são:

- Escolher entradas que gerem mensagens de erro.
- Escolher entradas que gerem o estouro de estruturas de armazenamento.
- Forçar a geração de saídas inválidas.
- Repetir séries de entradas.
- Escolher entradas limites de categorias de valores.
- Forçar a geração de valores muito grandes (pequenos).

A corretude do sistema é estabelecida a partir da análise dos resultados obtidos como saída do sistema, os quais devem ser inspecionados em relação aos resultados esperados, estabelecidos a priori. A partir desta análise é possível aferir o nível de qualidade atingida.

Atividades de avaliação

1. Defina cobertura de testes e explique por que é importante maximizar a cobertura na seleção dos casos de teste a serem executados.
2. Explique por que não é possível garantir a ausência de erros a partir da execução de processos de V & V.
3. Pesquisar por testes de regressão.
4. Explique a importância da automatização de testes e a utilização de frameworks de testes.
5. Pesquise por uma ferramenta de apoio à execução de testes.
6. Caracterize testes caixa preta e caixa branca.
7. Explique o diagrama em V dos testes.
8. A partir dos casos de uso especificados em exercícios anteriores descreva cenários de teste a serem contemplados na verificação do sistema.

Finalmente, uma vez que o sistema é testado na integridade e aceito pelo cliente, dá inicio a fase de Implantação envolvendo o empacotamento, distribuição e instalação do sistema no ambiente do usuário. Adicionalmente, é prevista a geração de manuais e treinamento.

2.3. A fase de manutenção

Uma vez que o sistema é entregue ao cliente e colocado em operação dá inicio a fase de manutenção ou evolução do software. Considerando os altos custos envolvidos na aquisição de um produto de software, este representa um investimento para a organização a ser utilizado por certo período de tempo, enquanto se mostre útil no seu ambiente.

Mais do 50% de todo o esforço investido em desenvolvimento é gasto em manutenção, e esta percentagem continua a crescer na medida em que mais software é produzido. Isto se deve a que, de forma geral, o software do qual dependemos tem de 10 a 15 anos de idade, e ao longo desse tempo todo foi preciso se adaptar às mudanças no ambiente e às constantes demandas dos usuários. Com isso, a pesar do sistema ter sido desenvolvido inicialmente utilizando boas práticas de análise e projeto (e isto nem sempre acontece), a qualidade da arquitetura global pode ter sido prejudicada como consequência de manutenções mal projetadas. A fase de manutenção focaliza nas mudanças que ocorrerão depois que o software for liberado para uso operacional.

No inicio, normalmente acontecem mudanças vinculadas à correção de erros que não foram detectados durante o desenvolvimento. Este tipo de manutenção algumas vezes é prevista durante um período de tempo estabelecido contratualmente.

Considerando que os sistemas de software são fortemente acoplados a seu ambiente, quando o sistema é instalado em um ambiente que muda, novas adaptações podem ser necessárias de forma a manter o sistema aderente.

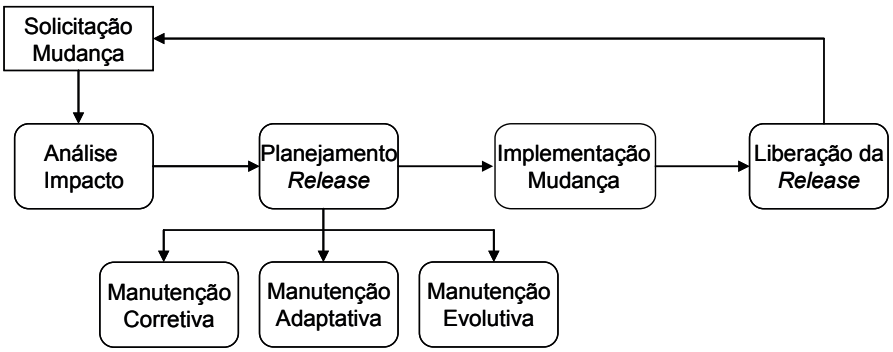
Geralmente as mudanças são implementadas modificando componentes existentes e/ou adicionando novos componentes ao sistema. Na fase de manutenção os passos das fases de definição e desenvolvimento são reaplicados, mas agora no contexto de um software existente.

A atividade de manutenção pode ser classificada em:

- Manutenção para reparar defeitos do software (corretiva), que visa modificar o sistema para corrigir deficiências em relação aos requisitos. Em geral este tipo de manutenção não envolve mudanças relevantes na arquitetura do sistema.
- Manutenção para adaptar software (adaptativa) visa modificar o sistema para operar em um ambiente diferente da sua implementação original.
- Manutenção para adicionar ou modificar a funcionalidade do sistema (evolutiva) envolve modificar o sistema para satisfazer novos requisitos.

Estudos mostram que aproximadamente o 50% do tempo invertido em manutenção é gasto no entendimento do código. Neste contexto, a documentação tem um papel fundamental.

Todo processo de manutenção deve ser embutido no contexto de um processo de mudança onde, a partir de uma solicitação de mudança é realizada a análise do impacto que essa mudança pode acarretar no sistema. A partir dessa avaliação, onde os aspectos de tempo e custo envolvidos precisam ser considerados, a mudança pode ser aprovada ou não, justificando devidamente a decisão tomada. No caso da solicitação ser aceita, o planejamento das atividades envolvidas na implementação da mudança é realizado, estimando em que momento a nova versão do sistema será liberada (release).

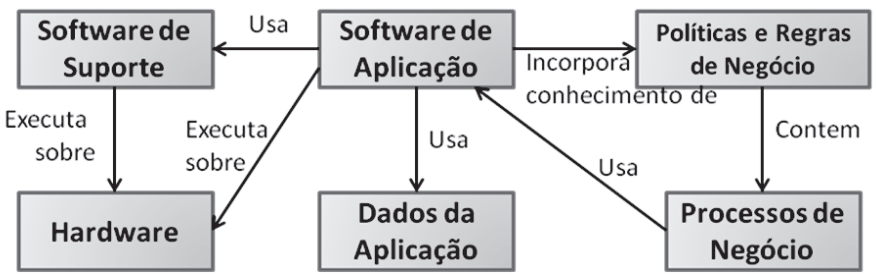


Um fator que tem uma importante influência no custo da manutenção, além da estabilidade e experiência da equipe é a idade e estrutura dos programas. Quanto mais antigos os programas forem, se pressupõe que tenha passado por um número maior de mudanças e atualizações. Dependendo da qualidade com que foram realizadas essas manutenções ao longo do tempo, a estrutura dos programas pode ter sido degradada, tornando cada vez mais complexas e onerosas as mudanças.

2.4. Sistemas legados e reengenharia

O crescente problema da manutenção se torna crítico para o caso de **sistemas legados**. Sistemas legados são sistemas de software antigos que foram desenvolvidos sob encomenda para uma organização, faz tempo, mas que ainda são fundamentais para o normal funcionamento da mesma.

Sistemas legados são sistemas socio-técnicos baseados em computador, que incluem: software, hardware, dados e processos corporativos.



Sistemas devem mudar para se manter úteis, no entanto a incorporação de mudanças em sistemas legados pode ser dispendiosa e difícil. A causa principal desta dificuldade é que ao longo das diversas manutenções realizadas, a estrutura do sistema pode ter sido corrompida. Isto acontece por diversos motivos, em particular a falta de uma especificação completa e atualizada do sistema dificulta seu entendimento. Com isso, o projeto das mudanças a serem realizadas pode ser dificultado: a carência de uma visão

global da arquitetura do sistema faz com que as mudanças sejam feitas de forma conservativa no intuito de minimizar o impacto no restante do sistema.

Outro fator que dificulta o entendimento do sistema é que geralmente não há um estilo de programação consistente e uniforme em todos os programas, assim como também a utilização de "truques" de programação.

Similarmente ao que acontece com os programas, a estrutura e integridade das informações pode estar comprometida no caso de sistemas legados: dados disseminados ao longo de diferentes arquivos podem acarretar na possível incompatibilidade, replicação e inconsistência de informações, levando a erros ou falhas na execução do sistema.

Por outro lado, a tecnologia que foi utilizada no desenvolvimento de sistemas legados pode ser atualmente obsoleta, dificultando a procura por mão de obra habilitada para fazer a manutenção.

No entanto, e a pesar destas dificuldades, os sistemas legados podem ainda ser críticos para a organização, isto é, essenciais para o normal funcionamento do negócio. Diante deste cenário a organização pode optar por continuar a manter o sistema através da manutenção normal, substituir o sistema legado por um novo sistema ou submeter o sistema a um processo de rejuvenescimento ou reengenharia.

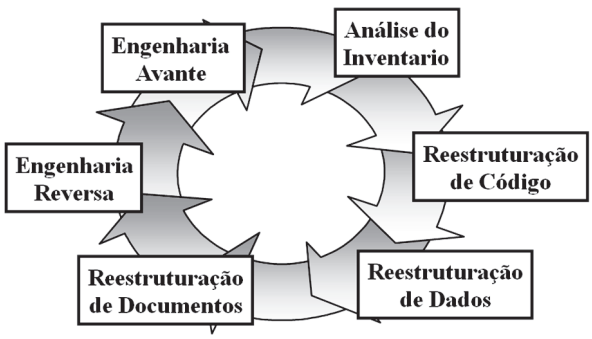
Considerando que normalmente a manutenção tradicional se torna cada vez mais custosa no caso de sistemas legados, esta opção deixa de ser viável.

A substituição de um sistema legado por outro pode envolver altos riscos para a organização, uma vez que todo novo empreendimento envolve riscos. Isto acontece por que o sucesso e a plena satisfação do cliente nem sempre podem ser garantidos. Esta afirmação é particularmente verdadeira no caso do novo sistema vir a substituir um sistema legado. Considerando que os processos corporativos e o modo como os sistemas legados operam estão sempre intrinsecamente entrelaçados, importantes regras corporativas estabelecidas ao longo do tempo podem estar inseridas no software e podem não estar documentadas em outro lugar. Consequentemente, a migração para um sistema mais moderno pode acarretar em perda de informação crucial para a organização.

Neste contexto, surge a reengenharia de software com o objetivo de reimplementar sistemas legados de forma a facilitar a sua manutenção reduzindo os custos envolvidos. A reengenharia é fundamentada em técnicas da Engenharia de Software visando ampliar a vida útil dos sistemas legados no intuito prolongar a vida útil do sistema a um custo viável. A estratégia utilizada nos processos de reengenharia envolve a re-estruturação ou re-escrita de parte ou todo de um sistema legado, mas sem mudar suas funcionalidades. Considerando que um sistema legado é construído ao longo de anos de uti-

lização, nem sempre são todos os programas que o integram que precisam passar por processos de reengenharia. De forma geral é aplicável quando algum, mas não todos os subsistemas de um sistema maior, precisam de frequente manutenção.

A reengenharia é baseada em sub-processos que objetivam reestruturar e redocumentar o sistema. A figura a seguir ilustra os processos da reengenharia.



A partir da análise do inventário de todas as aplicações envolvidas é gerada uma planilha com a descrição detalhada de cada aplicação ativa, ordenada segundo algum critério. Normalmente o critério utilizado consiste em uma função que depende da qualidade global do sistema e a importância que o sistema tem para a organização. A partir desta planilha são estabelecidas as aplicações candidatas a passar pelo processo de reengenharia.

A reestruturação de código ou documentos é um sub-processo da reengenharia praticado com frequência, que consiste em modificar os artefatos de software em um esforço por torná-los mais amenos e consistentes para seu uso em manutenções futuras, mas sem modificar a arquitetura global do programa. A reestruturação pode acontecer nos níveis de documentação, código e dados. Em alguns casos, o processo de reestruturação é seguido de um processo de tradução envolvendo a transformação do sistema de uma representação para outra no mesmo nível de abstração relativo, preservando o comportamento funcional externo.

A engenharia reversa é um processo mais complexo e caro que envolve uma abordagem de engenharia para o entendimento, análise e abstração do sistema legado a um novo formato em um nível mais alto de abstração. O objetivo deste processo é recuperar o projeto e especificação do sistema a partir da análise do software, identificando componentes e relacionamentos e gerando representações do sistema em um nível mais alto de abstração do que o código fonte. Este processo é particularmente útil quando o sistema carece de documentação ou esta se encontra muito desatualizada. Como resultado do processo é construída uma base de dados do programa a partir da qual pode ser construído conhecimento sobre o sistema.

Finalmente, a engenharia progressiva ou avante (forward) consiste em um conjunto de atividades de engenharia que utiliza como base os produtos e artefatos derivados de sistemas legados para a inclusão de novos requisitos. Produtos gerados na engenharia reversa de legados podem alimentar a engenharia progressiva de um novo sistema.

Saiba mais

Quantidade de código legado...

Em 1990 foram estimadas 120 bilhões de linhas de código em manutenção.

Em 2000 foram 250 bilhões de linhas de código sendo mantido, e continua crescendo. A proporção de código legado nas organizações aumenta em 10% a cada ano.

A quantidade de código em manutenção duplica a cada 7 anos.

Antigas linguagens não morreram... Ex. 70% das aplicações de negócio ativas estão escritas em COBOL .

Existem pelo menos 200 bilhões de linhas de código COBOL em computadores mainframe.

Atividades de avaliação

1. Quais são as estratégias comuns de mudanças de software?
2. Descreva os tipos de manutenção mais comuns.
3. Por que o custo da munutenção é maior que os custos de desenvolvimento?
4. Quais fatores influenciam os custos de manutenção?
5. Defina sistema legado. Explique a sua importância para o funcionamento de uma organização.
6. De que depende a estratégia de evolução a ser utilizada para um determinado sistema legado?
7. Explique a relação de dependência entre processos de negócio da organização e o sistema. Justifique por que no caso de sistemas legados esta dependência pode ser crítica.
8. Pesquise por um dos sub-processos de reengenharia mencionados no texto.

3. Modelos de Ciclo de vida de software

Na busca por atingir uma maior qualidade nos produtos de software gerados, engenheiros de software têm focado seus esforços no processo de desenvolvimento, reconhecendo que a organização e disciplina com que são realizadas as atividades contribuem para a qualidade do produto resultante. No entanto, diferentes tipos de sistemas de software podem requerer a adoção de diferentes tipos de processos, dependendo das características do produto em questão. Por exemplo, sistemas pequenos que podem ser desenvolvidos por uma equipe reduzida para um domínio de aplicação conhecido, podem requerer de um rigor menor na execução de atividades de planejamento e controle do que um sistema de porte maior. Sistemas maiores normalmente envolvem maior complexidade, investimento maior e de uma equipe de desenvolvimento maior para um domínio de aplicação não trivial, cujo desenvolvimento pode se estender por um longo tempo. Com isso se torna necessário um rigor maior nas atividades gerenciais de planejamento, monitoramento e controle das atividades.

O encadeamento específico das atividades envolvidas no processo de desenvolvimento de um sistema determina o ciclo de vida adotado, descrevendo a vida do produto desde a sua concepção até entrega e manutenção.

A adoção de um processo para o desenvolvimento orienta as ações a serem seguidas, possibilitando entender, analisar, monitorar e melhorar a realização das atividades que o compõem, facilitando a transmissão de experiências e lições aprendidas.

A escolha de um Ciclo de Vida de Software é possivelmente uma das primeiras decisões de projeto que precisam ser resolvidas pelo engenheiro de software uma vez que determina a abordagem sistemática que será seguida para desenvolver um projeto de software. Esta escolha depende dos seguintes fatores:

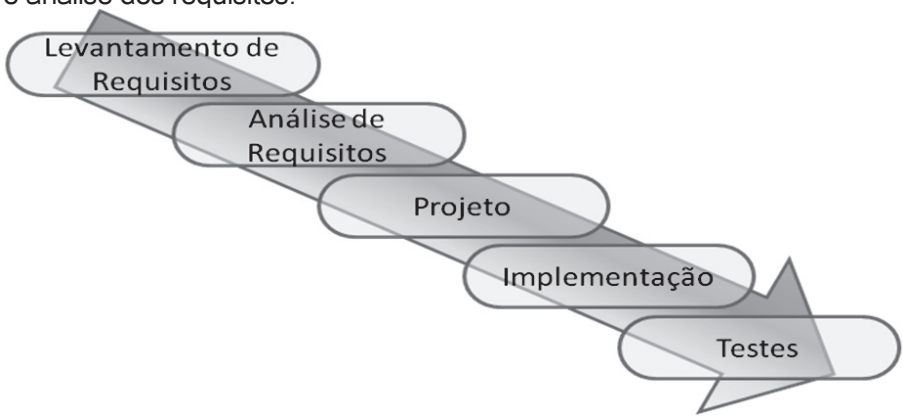
- Natureza do projeto e da aplicação.
- Métodos e ferramentas a serem usados ou disponíveis.
- Controles requeridos e produtos a serem entregues.
- Prazos e recursos disponíveis.

Nas seções a seguir são apresentados os modelos de ciclo de vida mais difundidos na literatura.

3.1. Ciclo de vida tradicional ou cascata

No ciclo tradicional, o fluxo adotado para a execução das atividades relativas ao desenvolvimento do produto de software segue uma ordem sequencial, onde uma atividade somente é iniciada uma vez que a anterior for concluída na sua totalidade. A partir da premissa de que toda atividade é executada na sua totalidade, iterações no ciclo não são previstas.

Este modelo é simples e de complexidade baixa no gerenciamento uma vez que não é prevista a execução de atividades em paralelo, e se mostra adequado para projetos de pequeno ou médio porte, cujo domínio de aplicação é conhecido para a equipe. No entanto, para projetos que fogem a estas características, isto é, de grande porte em termos de escopo, duração e orçamento, a adoção do modelo tradicional pode não atingir o patamar de qualidade esperado. Outro aspecto que influencia é o grau de conhecimento sobre o domínio da aplicação. Neste caso, a realização completa e satisfatória das atividades tem mais chance de sucesso, como por exemplo, o levantamento e análise dos requisitos.



Este modelo apresenta várias desvantagens, algumas das quais são listadas a seguir:

- É difícil estabelecer explicitamente todos os requisitos no início. No começo dos projetos sempre existe uma incerteza natural que é gerenciada ao longo do desenvolvimento do produto, na medida em que o conhecimento sobre o domínio é construído.
- Considerando a constatação de que os requisitos mudam com o decorrer do tempo, ao longo do processo de desenvolvimento. Estas mudanças precisam ser absorvidas assim que possível, caso contrário estes desvios podem envolver altos custos por conta do retrabalho, colocando em risco a continuidade do projeto.
- A versão executável do software só fica disponível em uma etapa avançada do desenvolvimento. Como consequência, o risco do sistema não atender completamente às necessidades do cliente fica latente até o final do desenvolvimento. Problemas decorrentes desta situação dificilmente poderão ser contornados de forma satisfatória em um estágio tão avançado do desenvolvimento.
- Projetos reais raramente seguem o fluxo sequencial que o modelo propõe.

O modelo tradicional ainda continua a ser utilizado, no entanto, a partir das deficiências constatadas, outros modelos de ciclo de vida têm sido propostos.

3.2. Ciclos Iterativos

Em resposta à constante evolução e permanente aumento da complexidade dos sistemas, o modelo de desenvolvimento iterativo surge como uma solução eficaz para lidar com as exigências impostas pelo mercado de aplicações de software. Em particular, a espera por longos períodos de tempo entre a especificação do sistema e a sua entrega efetiva não são mais toleráveis nos tempos atuais.

Mudança são inevitáveis, principalmente em projetos de grande porte. As mudanças podem ser originadas em resposta a pressões internas ou externas à organização, mudança nas prioridades, surgimento de novas tecnologias, etc. Neste contexto, é difícil, ou mesmo impossível, estabelecer a priori uma especificação detalhada dos requisitos. A ocorrência de mudanças implica em que partes do sistema em desenvolvimento devem ser re-trabalhadas. Desta forma, atividades de processo precisam ser repetidas em resposta às mudanças solicitadas.

O principio que guia os processos iterativos é que ao longo de cada iteração é desenvolvido o software e a especificação em forma conjunta. Esta abordagem flexibiliza a incorporação de mudanças ao longo do processo de desenvolvimento, uma vez que no fim de cada iteração, o planejamento da próxima pode absorver rapidamente possíveis mudanças, reduzindo o impacto das mesmas.

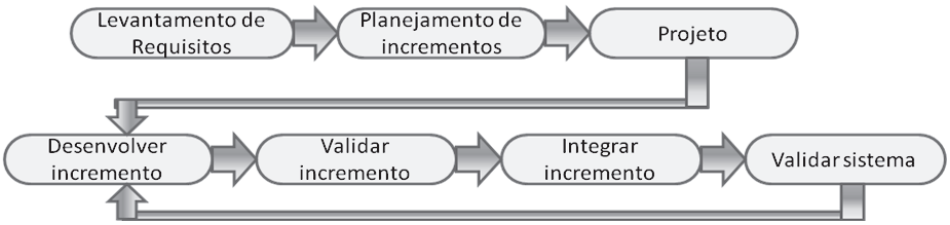
Em contrapartida, esta abordagem pode acarretar alguns problemas que, dependendo da política seguida pela organização contratante, podem ser inconciliáveis. Algumas das dificuldades encontradas durante o desenvolvimento utilizando a abordagem iterativa são:

- Problemas de gerenciamento devido às dificuldades de gerenciar as diversas iterações ocorrendo em forma simultânea.
- Problemas de contrato e validação. Normalmente, a especificação do software a ser desenvolvido é utilizada como base para cálculo de estimativas utilizadas como fundamentação para a assinatura do contrato entre as partes. Sem uma especificação completa do sistema, os clientes podem ficar receosos.
- Problemas de manutenção, uma vez que a concepção inicial do sistema pode se ver corrompida por constantes mudanças, dificultando seu entendimento e evolução.

Iterações podem ser aplicadas a qualquer modelo de processo genérico. Em particular, o modelo iterativo incremental combina elementos do modelo cascata com a filosofia iterativa da **prototipação**. O ciclo incremental propicia a facilidade de incorporação de mudanças, inerente aos modelos iterativos, mas adicionalmente, focaliza na entrega rápida de funcionalidades ao

cliente, uma vez que o desenvolvimento e entrega do sistema é particionado em incrementos, onde cada incremento realiza uma parte da funcionalidade requerida. Este processo se repete ao longo das iterações até que o produto final é construído na sua totalidade.

Neste contexto, na fase de análise, os requisitos de usuário são priorizados, e com base nessa priorização é feito o planejamento das iterações, onde os requisitos de maior prioridade são considerados nos incrementos iniciais. Uma vez que um incremento é iniciado, os requisitos são congelados, no entanto, requisitos para incrementos posteriores podem continuar a evoluir. O núcleo do sistema desenvolvido nos incrementos iniciais passa a ser usados pelo cliente na medida em que são liberados. Com base no retorno dado pelo cliente a partir do uso do sistema e da adição de novas funcionalidades, o planejamento da próxima iteração é realizado.



A característica principal que diferencia o modelo incremental é que ao fim de cada incremento um produto operacional é entregue ao cliente para a respectiva validação. Como vantagens do desenvolvimento incremental temos as seguintes percepções:

- Uma entrega de valor para o cliente pode ser liberada em cada incremento, assim as funcionalidades do núcleo do sistema são alcançadas mais cedo.
- Incrementos iniciais atuam como um protótipo na busca de elicitar os requisitos de incrementos posteriores. Desta forma, serviços do sistema de prioridade mais alta tendem a receber mais testes parciais.
- Os riscos de falha sobre o projeto são reduzidos, uma vez que os riscos técnicos podem ser melhor administrados.

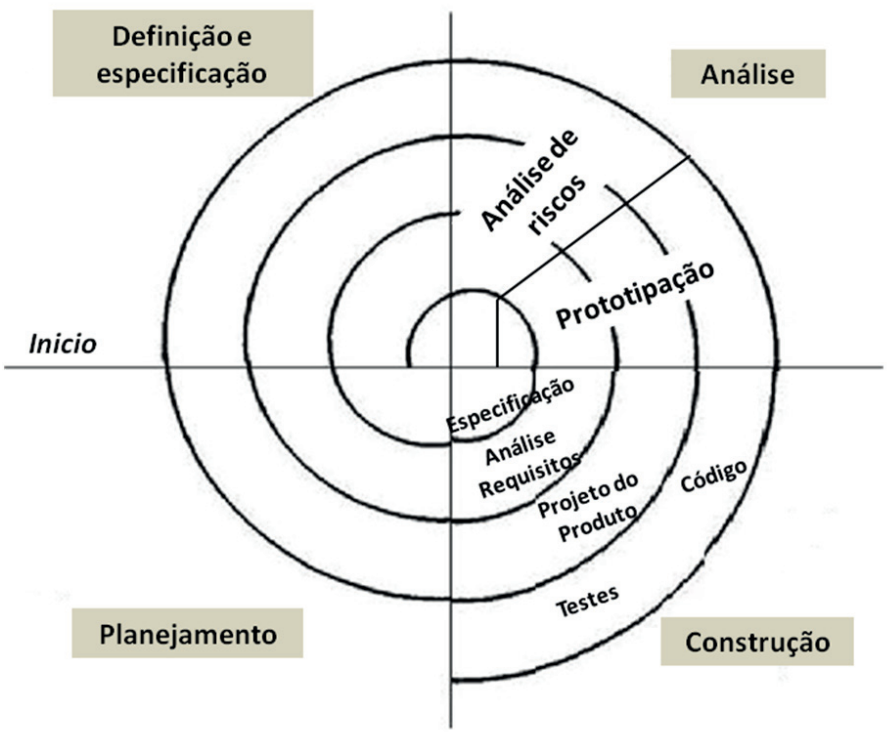
O desenvolvimento em espiral é um modelo iterativo, que também se enquadra no chamado **ciclo evolucionário**, uma vez que o produto é desenvolvido ao longo de uma sequência de versões cada vez mais detalhadas e completas, na medida em que seu grau de risco se torna cada vez mais baixo.

O desenvolvimento evolucionário se baseia na idéia de que o desenvolvimento do sistema acontece ao longo de uma sequência de versões, através das quais uma implementação inicial é refinada e completada a partir do retorno dado pelos clientes.

Problemas decorrentes da falta de uma especificação completa desde o inicio do desenvolvimento pode dificultar as atividades de planejamento uma vez que o número de ciclos para construir o produto é incerto. Adicionalmente, a incorporação frequente de mudanças durante o processo de refinamento podem corromper a estrutura do sistema, fazendo com que sua evolução e manutenção fiquem mais difíceis e onerosas.

Cada iteração na espiral representa uma fase de projeto que é dividida em setores, incluindo:

- Definição e especificação do problema. Nesta fase é definido um plano de gerenciamento com base na definição dos objetivos e nas restrições sobre o processo e o produto, realizando um levantamento dos riscos envolvidos.
- Análise. Com base nos riscos levantados é feita uma análise objetivando a definição de medidas para mitigar ou reduzir a incidência dos mesmos. O modelo prevê a utilização de protótipos como uma forma de redução de riscos. Estas técnicas permitem uma modelagem mais realística do domínio do problema.
- Construção. Nesta fase o produto é construído.
- Planejamento. O projeto é avaliado e com base nele é realizado o planejamento da próxima iteração na espiral.



No modelo espiral, as atividades de análise de riscos e prototipagem são obrigatórias em todos os estágios do projeto. Com isso, os riscos podem ser monitorados de perto, e ações preventivas e corretivas podem ser aplicadas de forma a minimizar seus impactos. Em contrapartida, a análise dos riscos é uma atividade difícil de ser realizada e requer muita experiência na descoberta e avaliação dos riscos.

Saiba mais

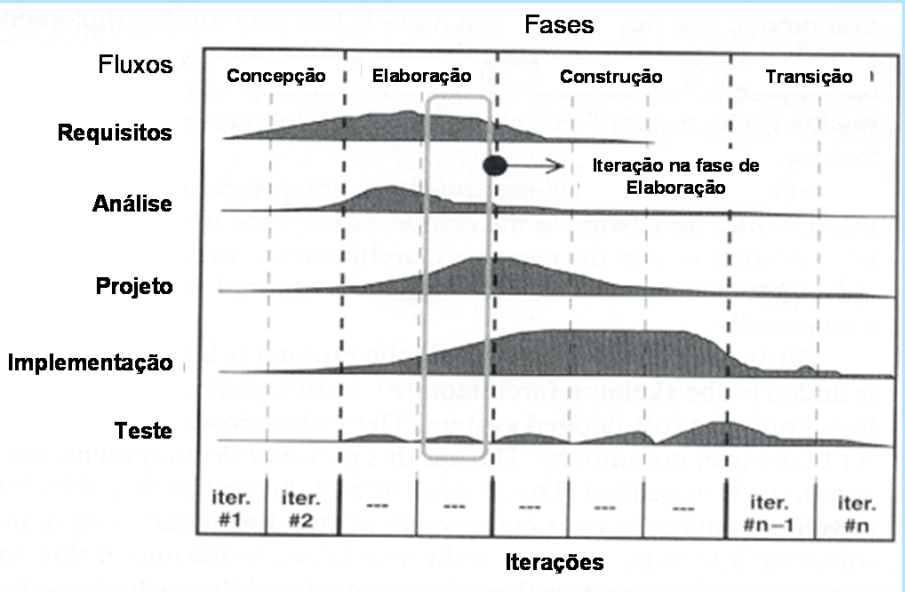
O Processo Unificado (PU)

O processo unificado (UP) é um modelo de ciclo de vida evolucionário de desenvolvimento de software que combina os ciclos iterativo e incremental para a construção de software. O PU consiste em um conjunto de atividades necessárias para transformar requisitos do usuário em um sistema de software. O processo unificado é caracterizado por três conceitos chave, a saber:

- direcionado a casos de uso;
- centrado na arquitetura;
- iterativo e incremental.

O PU preve a construção do sistema de software ao longo de iterações, incrementalmente, onde no final de cada iteração é gerada uma nova versão do produto.

O processo unificado utiliza a Linguagem de Modelagem Unificada (Unified Modeling Language – UML) no preparo de todos os artefatos do sistema.



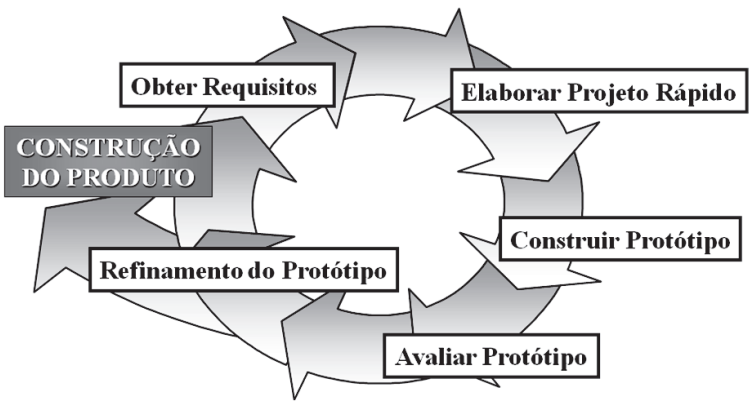
Produto da empresa de software Rational Software Corporation, o RUP representa uma instância específica e detalhada do Processo Unificado, que entre outras coisas, adiciona novas disciplinas ao PU original.

A prototipagem de software se baseia no desenvolvimento rápido de uma versão preliminar do software, ou protótipo, desenvolvido, principalmente, para entender melhor o problema junto com o cliente, na busca de possíveis soluções.

Idealmente, o modelo (protótipo) serve como um mecanismo para identificar os requisitos de software, e é especialmente útil quando o cliente definiu um conjunto de objetivos gerais para o software, mas não identificou requisitos de entrada, processamento e saída de forma detalhada.

O ciclo de prototipagem é iniciado estabelecendo junto ao cliente, os requisitos gerais do sistema, a partir dos quais é modelado com base na elaboração um projeto rápido. Considerando que o protótipo não necessariamente irá a modelar todas as funcionalidades do sistema, a escolha pelas funcionalidades a serem abordadas no protótipo precisam ser determinadas. Em particular, no projeto são considerados aspectos visíveis e de interesse do cliente, no intuito de construir uma visão mais completa do sistema a ser desenvolvido, a partir do retorno esperado dos clientes. A partir dessa modelagem, o protótipo é construído e apresentado ao cliente. O protótipo possibilita ao cliente ter uma visão mais concreta do sistema sendo desenvolvido, tornando possível experimentar e avaliar a viabilidade do projeto. Novas iterações podem vir a acontecer se houver necessidade de realizar ajustes no protótipo.

Embora o principal objetivo do protótipo seja entender os requisitos do usuário também pode ser usado para testar e demonstrar conceitos. Por exemplo, um protótipo pode ser gerado para testar a adequação de uma determinada tecnologia às necessidades do projeto. Desta forma, a prototipagem pode ser utilizada como uma técnica de engenharia aplicada dentro do contexto de qualquer modelo ou ciclo de desenvolvimento e como um mecanismo de mitigação de riscos.



A especificação dos requisitos obtida a partir da execução do ciclo de prototipagem irá alimentar as fases posteriores no processo de desenvolvimento do produto. Concluída a fase de prototipagem, e levantadas as informações necessárias, o protótipo deve ser descartado.

O descarte do protótipo pode ser problemático, tanto para clientes quanto para desenvolvedores. Por um lado, o cliente não aceita bem a idéia que a versão final do software ainda precisa ser construída, uma vez que o protótipo aparenta ser uma versão executável do produto, e pode estimular a utilização do protótipo como produto final. Isto acontece principalmente por que o cliente ou usuário desconhece que no protótipo não foram consideradas, durante o desenvolvimento, a qualidade global e a manutenibilidade de sistema em longo prazo.

No desenvolvimento do protótipo, o desenvolvedor frequentemente faz uma implementação comprometida com o objetivo de produzir rapidamente um modelo executável. Depois de um tempo ele se familiariza com essas escolhas, e esquece que elas podem não ser as mais apropriadas para o produto final, uma vez que os requisitos não funcionais não são ser levados em consideração nesta fase.

A utilidade da realização do ciclo de prototipagem no contexto de desenvolvimento de software é inegável, no entanto, é importante que a função do desenvolvimento do protótipo entre cliente e desenvolvedor fique claramente estabelecida: ambos devem concordar em que o protótipo é construído para servir como um mecanismo a fim de definir os requisitos e que, cumprido esse objetivo, precisa ser descartado. A geração de protótipos pode sugerir um aumento nos custos globais do projeto, no entanto, redução de re-trabalho e consequentes ganhos em produtividade compensam os custos nos estágios mais avançados de desenvolvimento.

Atividades de avaliação

1. Para cada tipo de modelo de processo explique como uma mudança significativa nos requisitos no final do desenvolvimento, pode implantar no processo.
2. Quais os benefícios e desvantagens de se utilizar cada tipo de modelo de processo descrito.
3. Explique por que o modelo incremental combina o modelo cascata com a filosofia iterativa da prototipação.
4. Descreva um exemplo de uso do modelo incremental aplicado ao desenvolvimento de um software para edição de textos, detalhando o planejamento da entrega de funcionalidades ao longo das iterações.
5. Dê três exemplos de projetos de software que seriam adequados para o modelo incremental.
6. Dê três exemplos de projetos de software que seriam adequados para o modelo espiral.
7. Dê três exemplos de projetos de software que seriam adequados para o modelo tradicional.
8. Explique de que forma os modelos apresentados podem ser combinados, descrevendo brevemente o fluxo de atividades envolvidas.
9. Explique por que o protótipo precisa ser descartado.
10. Uma das principais utilizações de protótipos é a modelagem de interfaces. Dê outros exemplos onde a utilização de protótipos pode ser interessante.

Síntese do capítulo

Nesta unidade foram apresentadas as principais atividades envolvidas na produção de um sistema de software. As fases foram organizadas em macro-atividades: definição, desenvolvimento e manutenção.

O encadeamento específico ou fluxo seguido para a execução destas atividades determina o processo de desenvolvimento de software ou modelo de ciclo de vida. As principais características dos modelos tradicional (cascata) e iterativos (incremental, espiral, prototipagem) foram apresentadas.

Adicionalmente, a importância do fator humano e o papel do engenheiro de software, suas atribuições e responsabilidades no cenário atual foram discutidos.

Leituras, filmes e sites

Livros

KENDALL, Scott. O Processo Unificado Explicado. Bookman, 2003.

BROOKS, Frederick P. No Silver Bullet Essence and Accidents of Software Engineering. Computer, vol. 20, num. 4, pp. 10-19. April, 1986.

OLSEM, M. Reengineering Technology Report. Vol I. TRF-RE-9510-00.04. 1995.

Booch, G.; Rumbaugh, J.; Jacobson, I. UML - Guia do Usuário. Campus, 2000.

Referências

PRESSMAN, R.S. Engenharia de Software. 5ª ed. Rio de Janeiro: McGraw-Hill, 2002.

SOMMERVILLE, I. Engenharia de Software. 6ª ed. São Paulo: Addison Wesley, 2003.

PFLEEGER, S. Engenharia de Software. Teoria e prática. 2da edição. Pearson, 2004.

PAULA FILHO e PÁDUA W. de. Engenharia de Software - Fundamentos, Métodos e Padrões. Ed. LTC, 2003.

Capítulo 3

Qualidade e sua garantia em projetos

Objetivos

- Esta unidade tem seu foco no principal conceito que orienta todas as atividades de engenharia: a qualidade. Fundamentado na adoção de normas e padrões, o gerenciamento da qualidade objetiva garantir que o produto de software desenvolvido como resultado de um projeto atinja o patamar de qualidade esperado, de forma a atender às necessidades dos interessados.

1. Introdução

Com a Revolução Industrial surgiram diversas iniciativas de elaboração de normas técnicas que estabeleceram padrões e parâmetros universais para determinados produtos e serviços, com o objetivo de beneficiar a cooperação e o intercâmbio desses produtos e serviços.

Na indústria, o projeto de um processo e de um produto é bastante complexo, pois exige múltiplas habilidades intelectuais e motoras, raciocínio, memória, dentre outras. Cada projeto deve ser feito cuidadosamente, pois um processo será repetido inúmeras vezes em diversos ciclos de produção, produzindo várias unidades do produto. Se o processo tiver sido mal- projetado, o prejuízo pode ser muito grande.

O estabelecimento de normas e padrões de qualidade tem incentivado as empresas à adoção de técnicas e tecnologia de apoio, de forma a tornar seus produtos mais competitivos. Os padrões representam um meio para atingir a satisfação dos clientes e de todos os envolvidos, de forma geral.

Com origem nos processos de manufatura, a atividade de gerenciamento da qualidade vem ganhando um papel de destaque no contexto do desenvolvimento de software. No entanto, a aplicação das práticas e princípios de gerenciamento de qualidade no contexto de produção de software é bem mais complexa que no caso de produtos de manufatura. Isto se deve às características inerentes ao produto sendo desenvolvido, uma vez que no caso de software, se trata de um produto intangível, difícil de ser especificado e medido.

A noção básica do conceito de qualidade está vinculada ao grau em que o produto desenvolvido atende à sua especificação. Entretanto, como discutido em capítulos anteriores, obter uma especificação completa do produto

de software orientada às características do produto é muito difícil. De forma geral, nem todos os aspectos que afetam e determinam o grau de satisfação dos clientes podem ter sido corretamente capturados na especificação. Adicionalmente, aspectos não-funcionais da qualidade de um produto podem ser difíceis de serem especificados de forma não-ambígua, por exemplo, os aspectos de adaptabilidade, facilidade de manutenção, eficiência, entre outros.

A adoção de padrões e procedimentos a nível organizacional é fundamental para o gerenciamento da qualidade, assim como também disseminar a cultura da qualidade no sentido de gerar um comprometimento da equipe com a qualidade dos trabalhos realizados na busca da melhoria contínua.

Fundada em 1940, a Associação Brasileira de Normas Técnicas (ABNT) é o órgão responsável pela normalização técnica no país, fornecendo a base necessária ao desenvolvimento tecnológico brasileiro. A ABNT tem como missão permitir a produção, a comercialização e uso de bens e serviços de forma competitiva e sustentável nos mercados interno e externo, contribuindo para o desenvolvimento científico e tecnológico, proteção do meio ambiente e defesa do consumidor.

Atendimento a requisitos especificados.

A ABNT é a única e exclusiva representante no Brasil das seguintes entidades internacionais: ISO (International Organization for Standardization), IEC (International Electrotechnical Commission), entre outras.

De acordo com a ABNT, qualidade pode ser definida como o grau no qual um conjunto de características inerentes satisfaz a requisitos. Neste contexto, uma característica é considerada uma propriedade diferenciadora e um requisito representa uma necessidade ou expectativa que é expressa, geralmente, de forma implícita ou obrigatória.

A qualidade de um produto pode ser avaliada considerando os seguintes aspectos:

- Técnicos. No âmbito do projeto, a qualidade está associada ao cumprimento das instruções de trabalho que regem o projeto, tais como: comunicação, cumprimento das metas, prazos e orçamento de cada pacote de trabalho, gerenciamento dos riscos, etc., conforme fora planejado.
- Funcionais. A solução deve agradar ao cliente, ser de fácil administração, robusta, confiável e estável.
- Documentação. Deve ser completa, clara, de fácil consulta e **aderente** aos padrões impostos pelo cliente.

No entanto, a qualidade atingida no produto desenvolvido depende diretamente da qualidade do processo que foi seguido para desenvolver o produto. Esta premissa é evidente nos processos de manufatura, onde uma vez que os processos são padronizados e calibrados, a monitoração garante a

execução contínua dos processos e a consequente construção de produtos de alta qualidade. Entretanto, por se tratar de uma atividade criativa, o desenvolvimento de software normalmente é influenciado pelas habilidades pessoais e experiência dos membros da equipe. Neste sentido, as ligações entre os aspectos de qualidade de processo e qualidade atingida no produto final são mais difíceis de serem estabelecidas no caso de um produto de software. Adicionalmente, aspectos externos tais como questões políticas e de mercado também podem influenciar na qualidade do produto, precipitando a liberação antecipada de uma nova versão driblando os controles de qualidade estabelecidos. No entanto, o aprimoramento na execução dos processos voltados a qualidade durante as atividades vinculadas ao processo de desenvolvimento tem uma influência comprovadamente positiva nos resultados obtidos.

Diferentemente do que todos acham, o software envolve não somente o conjunto de programas ou código fonte da aplicação, mas também toda documentação associada gerada durante o processo de desenvolvimento.

Considerando que a qualidade não acontece por acaso, se torna necessário esforço, investimento e apoio da alta direção da empresa, de forma a direcionar da melhor forma possível os seus esforços para atingir as metas de qualidade impostas.

O investimento em qualidade pode ser destinado em ações veiculadas no sentido de atingir a **conformidade** com foco na prevenção e avaliação, de forma a garantir o atendimento aos requisitos especificados. Por outro lado, o não atendimento ou **não conformidade** também envolve custos relativos a falhas internas e externas a serem reparadas, acarretando em re-trabalho. Os custos da não conformidade podem ser considerados como custos extras, consequência do produto ou serviço não ter sido feito corretamente na primeira vez. Um exemplo de custos envolvidos como consequência da não conformidade é o recall ou retirada do mercado de produtos que apresentaram defeitos ou falhas depois de terem sido liberados para uso. Com isso, é importante que o trabalho seja feito corretamente na primeira tentativa, economizando dinheiro e tempo. De forma geral, os custos dos reparos aumentam quanto mais demora a sua descoberta.

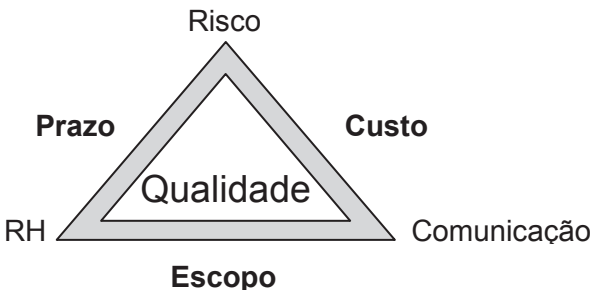
A qualidade é um processo preventivo que visa cumprir as exigências e especificações de modo a produzir produtos e serviços que atendam às necessidades do cliente, cujas funcionalidades foram devidamente testadas antes da entrega, prevenindo a ocorrência de falhas no produto entregue.

A qualidade é responsabilidade de todos os membros da equipe na busca pelo aprimoramento contínuo, e deve ser aplicada através de toda a organização. As premissas a serem levadas em conta para o gerenciamento eficiente e eficaz da qualidade incluem:

1. Satisfação do cliente
2. Prevenção sobre inspeção
3. Responsabilidade da gerencia
4. Melhoria contínua

A melhoria contínua de processos objetiva reduzir o desperdício e eliminar as atividades que não agregam valor, permitindo que os processos sejam operados com níveis mais altos de eficiência e eficácia.

Uma organização guiada pelos princípios da qualidade depende da definição e acompanhamento de processos específicos de gestão, não somente relativos ao produto final, mas também os relacionados à gestão de custos, prazo, e escopo que o produto deve atender com base na sua especificação preliminar e detalhada. O escopo pode ser interpretado como o conjunto de funcionalidades que o produto deverá atender em uso.



Adicionalmente, o gerenciamento dos riscos, recursos humanos e comunicação podem influenciar no grau em que os objetivos de qualidade serão atendidos.

Saiba mais

Qualidade à luz das normas ISO – International Organization for Standardization

A ISO é uma organização não-governamental fundada em 1947 em Genebra, cujo objetivo é promover o desenvolvimento de normas, testes e certificação com o intuito de encorajar o comércio de bens e serviços. A ISO hoje representa aproximadamente 150 paises, o que representa o 95% da produção industrial do mundo.

As Normas da Serie ISO 9000 formam um conjunto de normas relacionadas com gestão e garantia da qualidade (NBR ISO 9000).

Atividades de avaliação

1. Procure definições de qualidade diferentes ou complementares à apresentada nesta unidade.
2. Explique o relacionamento de dependência entre a qualidade do processo de desenvolvimento e a qualidade do produto resultante.
3. Explique a função da ABTN no contexto nacional e sua relação com a ISO.
4. Quais são as variáveis que influenciam na qualidade? Explique como acontece essa influência.

2. Normas e Padrões de qualidade

Os processos de garantia da qualidade (Quality Assurance) são fortemente baseados na adoção de padrões de qualidade. A adoção de padrões em todos os aspectos de desenvolvimento é uma boa prática uma vez que o estabelecimento de um padrão pressupõe um conhecimento aprimorado sobre as melhores e mais adequadas práticas para a empresa. Esse conhecimento surge após inúmeras tentativas e falhas. Com isso, uma vez estabelecidos, os padrões evitam cometer erros do passado e reduzem o esforço de aprendizado. A abordagem baseada em padrões precisa levar em consideração a premissa de que a qualidade no processo influencia na qualidade no produto, consequentemente padrões de projeto e de produto precisam ser levados em consideração.

Os padrões de produto se aplicam diretamente sobre o produto em desenvolvimento, enquanto que padrões de processo focam nas atividades que devem ser seguidas durante o desenvolvimento. De forma geral estes padrões são dependentes no sentido em que os padrões de processo precisam garantir que os padrões de produto possam ser atendidos.

As equipes de garantia da qualidade que desenvolvem padrões para uma organização, devem tomar como base padrões nacionais e internacionais, os quais precisam ser adequados à realidade e **cultura da organização.** A fim de auxiliar as empresas no gerenciamento de qualidade através da padronização dos processos, foram criados modelos baseados em normas internacionais e em boas práticas de engenharia de software. Infelizmente, ainda existem algumas empresas que não possuem a cultura da qualidade e não adotam esses modelos.

Um aspecto que desestimula às empresas a não terem se interessado pelo investimento em modelos que padronizam processos de desenvolvimento de software é que a certificação custa caro e as empresas têm pouca conscientização por processos.

Instituições nacionais e internacionais têm sido ativas na especificação, divulgação e estabelecimento de padrões. Exemplos de organizações são: US DoD (United States Department of Defense), ANSI (American National Standards Institute), BSI (British Standards Institution), NATO (North Atlantic Treaty Organization) e a IEEE (Institute of Electrical and Electronic Engineers).

2.1. A gestão da qualidade total e o ciclo PDCA

A gestão da qualidade total (Total Quality Management ou TQM) consiste em uma estratégia de administração orientada a criar consciência da qualidade em todos os processos organizacionais, envolvendo todos os escalões de uma

A certificação permite avaliar as conformidades determinadas pela organização através de processos internos, garantindo ao cliente um material, processo, produto ou serviço concebido conforme padrões, procedimentos e normas.

organização, além dos seus fornecedores, distribuidores e demais parceiros de negócios. Todos os interessados são diretamente responsáveis pela consecução dos objetivos da organização. Desse modo, a comunicação organizacional, em todos os níveis, torna-se uma peça-chave da dinâmica da organização.

A conscientização e a busca da qualidade e do reconhecimento da sua importância tornaram a **certificação** dos sistemas de gerenciamento da qualidade indispensável uma vez que:

- Aumenta a satisfação e a confiança dos clientes;
- Aumenta a produtividade;
- Reduz os custos internos;
- Melhora a imagem e os processos de modo contínuo;
- Possibilita acesso mais fácil a novos mercados, aumentando a amplitude da organização.

A melhor forma de garantir a sobrevivência de uma empresa é por meio da qualidade, entendida não como ausência de defeitos, mas como uma nova forma de valores que conduz a gestão. Ela aponta para a preferência do consumidor, o que aumenta a produtividade, levando a uma maior competitividade e assegurando a sobrevivência das empresas.

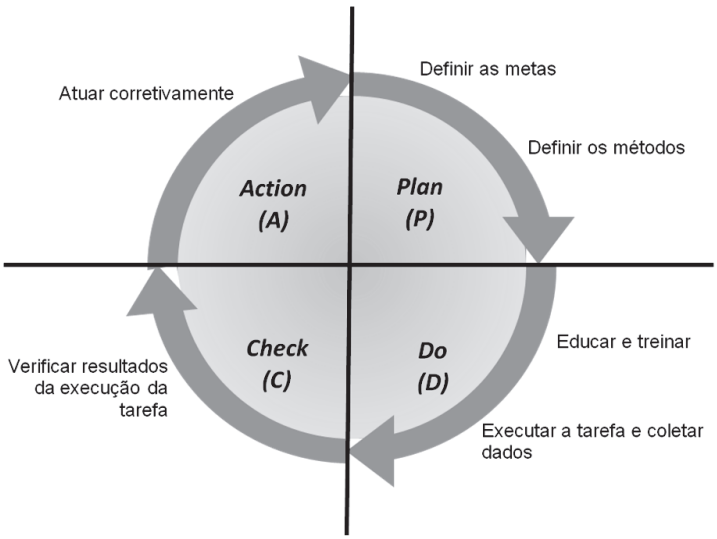
A qualidade total consiste na busca da satisfação, não só do cliente, mas de todos os stakeholders e também da excelência organizacional da empresa através da qualidade. Os princípios básicos da qualidade total são:

O PDCA foi introduzido no Japão após a guerra, idealizado por Shewhart e divulgado por Deming.

- Produzir bens ou serviços que respondam concretamente às necessidades dos clientes.
- Garantir a sobrevivência da empresa por meio de um lucro contínuo obtido com o domínio da qualidade;
- Identificar o problema mais crítico e solucioná-lo pela mais elevada prioridade;
- Falar, raciocinar e decidir com dados e com base em fatos;
- Administrar a empresa ao longo do processo e não por resultados;
- Reduzir metodicamente as dispersões por meio do isolamento das causas fundamentais;
- A prevenção deve ser a tão montante quanto possível;
- Nunca permitir que um problema se repita.

A lógica para que as empresas consigam se desenvolver de acordo com estes pressupostos é a lógica do **PDCA** (Plan; Do; Check; Act to correct). O ciclo PDCA é um ciclo de desenvolvimento que tem foco na melhoria contínua e pode ser descrito resumidamente como segue:

- *Plan* (Planejar): estabelecer os objetivos e processos necessários para atingir os resultados de acordo com os requisitos do cliente e com as políticas da organização e elaborar um plano de ação.
- *Do* (Fazer): implementar os processos conforme ao plano elaborado.
- *Check* (Verificar): monitorar e medir os processos e o produto frente ao planejado, às políticas, objetivos e requisitos para o produto e reportar os resultados.
- *Act* (Agir): agir de acordo com o avaliado e tomar ações para melhorar continuamente o desempenho do processo através da confecção de novos planos de ação, aprimorando a execução e corrigindo eventuais falhas.



O ciclo começa pelo planejamento, em seguida a ação ou conjunto de ações planejadas são executadas, checa-se se o que foi feito se encontra de acordo com o planejado, constantemente e repetidamente (ciclicamente), e toma-se uma ação para eliminar ou ao menos mitigar defeitos no produto ou na execução das atividades.

O PDCA é aplicado para se atingir resultados dentro de um sistema de gestão e pode ser utilizado em qualquer empresa de forma a garantir o sucesso nos negócios, independentemente da área de atuação da empresa.

2.2. *International Organization for Standardization* (ISO)

A família de normas ISO 9000 é reconhecida pelos consumidores e pelos empresários como sinônimo de qualidade. Este conjunto de normas estabelece um modelo de gestão de qualidade para qualquer tipo de organização, desde manufatura até prestação de serviços. A partir dela, surgiram diversas normas

e metodologias, entre elas a norma britânica BS570 produzida pela de British Standards Institute (BSI). As normas ISO reduziram o foco na necessidade de procedimentos valorizando mais a melhoria dos processos internos e os produtos. As normas são genéricas e podem ser aplicadas a domínios diversos, em contrapartida, cada organização precisa definir seu próprio manual da qualidade organizacional, documentando os procedimentos específicos adequados à realidade da organização, mas sem deixar de atender os requisitos da norma.

A série é composta pelas seguintes normas:

- ISO 9000 - Fundamentos e vocabulário.
- ISO 9001 - Sistemas de gerenciamento da qualidade - Requisitos.
- ISO 9004 - Sistemas de gerenciamento da qualidade - guia para melhoramento do desempenho.
- ISO 9011 - Auditorias internas da qualidade e ambiental.

A ISO no Brasil é representada pela Associação Brasileira de Normas Técnicas (ABTN). A série ISO 9000 foi adotada no Brasil com o nome de NBR 9000.

As organizações que desejam adotar a norma são avaliadas por uma equipe de auditores externa de forma a determinar a aderência à norma em relação aos procedimentos e operações de qualidade da empresa. Se for bem sucedida, a empresa recebe um certificado que comprova a sua qualidade. Os benefícios obtidos a partir da certificação estão relacionados ao aumento da confiabilidade por parte dos clientes, aumentando desta forma a sua competitividade no mercado. Posteriormente, auditorias externas periódicas são realizadas na empresa de forma a garantir a manutenção da conformidade com a norma. Dentre os benefícios esperados pela implantação das normas ISO, temos:

Do ponto de vista da organização:

- Participação competitiva no mercado.
- Maior satisfação dos clientes.
- Redução de custos e, consequemente, aumento do lucro.
- Melhoria na produção.

Já do ponto de vista dos clientes:

- Maior confiança e satisfação nos produtos da organização.
- Redução de custos.
- Melhor atendimento em caso de reclamações.

A norma ISO 9001 é de particular importância, uma vez que define os requisitos para a implantação do sistema de gerência de qualidade na empresa. Em outras palavras, esta norma define os requisitos a serem alcançados

para a certificação da empresa perante a ISO. Esta certificação não significa necessariamente, conformidade de produto às suas respectivas especificações, mas a capacidade da empresa de prover produtos que atendam aos clientes e aos requisitos regulatórios. A ISO 9001 é formada por requisitos:

1. Responsabilidade da administração.
2. Sistema da qualidade.
3. Análise crítica de contrato.
4. Controle de projeto.
5. Controle de documentos e dados.
6. Aquisição.
7. Controle de produto fornecido pelo cliente.
8. Identificação da rastreabilidade do produto.
9. Controle de processo.
10. Inspeção e ensaios.
11. Controle de equipamentos de inspeção e medição.
12. Situação de inspeções e ensaios.
13. Controle de produto não conforme.
14. Ação corretiva e preventiva.
15. Manuseio, armazenagem, preservação e entrega.
16. Controle de registros da qualidade.
17. Auditoria interna da qualidade.
18. Treinamento.
19. Serviços associados.
20. Técnicas estatísticas.

A ABNT NBR ISO 9001 é a versão brasileira da norma internacional ISO 9001 que estabelece os requisitos para o Sistema de Gestão da Qualidade (SGQ) de uma organização.

Atividades de avaliação

1. Caracterize a abordagem de qualidade total.
2. Explique como funciona o ciclo PDCA e ressalte seu principal objetivo.
3. Por que a adoção de padrões e normas pode ser considerada uma boa prática?

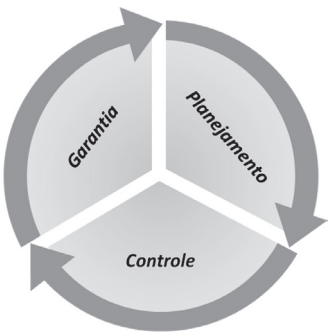
4. Consigne o papel e objetivos da ISO.
5. O que significa uma empresa ser certificada na adoção de uma norma?
6. Analise as vantagens da certificação e estabeleça as possíveis desvantagens.
7. Pesquise e gere um relatório sobre as normas ISO 9001.
8. Descreva brevemente padrões que podem ser adotados para:

   a) Escrita de programas.

   b) Solicitação e aprovação de mudanças.

   c) Reuso de componentes.

3. Os processos da qualidade



O gerenciamento de qualidade de software no desenvolvimento de sistemas de informação é baseado em três processos fundamentais: garantia de qualidade, planejamento de qualidade e controle de qualidade.

O gerenciamento da qualidade envolve as atividades da organização executora que determinam responsabilidades, objetivos e políticas de qualidade, de modo que o projeto atenda às necessidades que motivaram sua realização.

3.1. Planejar a Qualidade

O objetivo do planejamento da qualidade é garantir que as expectativas dos clientes sejam satisfeitas na medida necessária. Para alcançar este objetivo é preciso determinar e analisar as características do produto ou serviço, comparando-os às necessidades explícitas e implícitas dos interessados para entender como se correlacionam. O atendimento a este objetivo envolve a identificação de padrões e o estabelecimento das ações para satisfazê-los. Estão no contexto do planejamento da qualidade:

- Fornecer o produto com instruções claras sobre a utilização.
- Fazer a entrega pontualmente e dentro do custo orçado.
- Documentação de todos os processos envolvidos no gerenciamento do projeto.
- Identificar as normas e padrões que o projeto deve seguir, tais como ABNT, BS 7799, UML e outros, e determinar as estratégias de forma a satisfazê-los.

O processo de Planejar a Qualidade utiliza como entrada diversas informações relativas ao projeto tais como: descrição do projeto, principais entregas e critérios de aceitação, lista de interessados, o desempenho previs-

to para custos e cronograma e o registro de riscos. Por outro lado, **fatores ambientais da empresa** também precisam ser levados em conta de forma a adequar o planejamento conforme as políticas e normas adotadas pela organização. **Ativos de processo** relativos a projetos anteriores podem ser de grande utilidade no sentido de reduzir o re-trabalho e aumentar a produtividade da equipe, diminuindo os riscos.

A partir de uma aprimorada análise de custo-benefício fundamentada a partir de gráficos, experimentos e amostragem estatística, é gerado como resultado principal do processo o **Plano de gerenciamento** da qualidade. Este plano descreve como a equipe de gerenciamento do projeto irá implementar sua política de qualidade levando em conta os requisitos genéricos de qualidade relativos a normas e padrões, em requisitos específicos adequados a um produto ou projeto.

Os ativos de processo incluem políticas e procedimentos organizacionais de qualidade, dados históricos e lições aprendidas de projetos anteriores.

O plano pode ser redigido de forma informal ou estruturada, no entanto, alguns aspectos que não podem deixar de ser descritos são:

- Descrição do produto
- Planejamento de liberações e entregas.
- Descrição do processo e atividades envolvidas no desenvolvimento do produto.
- Metas de qualidade a serem atingidas, envolvendo a definição dos **atributos de qualidade do produto.**
- Riscos que podem influenciar a qualidade e ações que podem ser realizadas para minimizar seus efeitos.

Os atributos de qualidade a serem observados em um software são diversos, e geralmente são diretamente ligados à natureza da aplicação. Estes atributos orientam o processo de planejamento, e devem ser definidos e priorizados, uma vez que dificilmente o atendimento a todos os atributos poderá ser otimizado da mesma forma. Dependendo do tipo da aplicação, um atributo pode ser mais importante do que outro. Alguns atributos comumente vinculados a qualidade do software são: segurança, robustez, facilidade de manutenção, eficiência, facilidade de reuso, facilidade de aprendizado, entre outros.

O plano também deve definir algumas atividades extra, pertinentes à implementação da política de qualidade, no gerenciamento do escopo, custos e prazo. O plano aborda aspectos de Controle da qualidade, Garantia da qualidade e Melhoria contínua dos processos do projeto, indicando quando, como e por quem em relação a cada uma das atividades envolvidas.

Outros resultados obtidos a partir do processo de planejamento da qualidade são: **métricas de qualidade,** listas de verificação de qualidade, plano de melhorias no processo, e atualizações nos documentos do projeto.

Uma métrica de qualidade é uma definição operacional que descreve um atributo do projeto ou do produto, e como será medido no contexto do processo de controle da qualidade. A medição é um valor real, enquanto que a tolerância define as variações aceitáveis nas métricas.

As métricas estabelecidas no planejamento são utilizadas nos processos de garantia da qualidade e de controle da qualidade. Alguns exemplos de métricas de qualidade incluem: desempenho dentro do prazo, controle do orçamento, frequência de defeitos, taxa de falha, disponibilidade, confiabilidade e cobertura de testes.

1. Quais são os processos da qualidade?
2. Estabeleça os objetivos do processo de Planejar a qualidade.
3. Qual é o principal artefato gerado como resultado do processo de planejamento? Para que serve?
4. Realize uma pesquisa bibliográfica sobre métricas de qualidade.
5. Cite atributos comumente vinculados a qualidade do software.
6. Estabeleça os atributos de qualidade para os seguintes produtos de software:
   a) Aplicação de controle de decolagens e pousos em aeroporto.
   b) Software embarcado em dispositivos de lombada eletrônica.
   c) Aplicação bancária.
   d) Aplicação de ensino a distância para alunos do ensino fundamental.
   e) Sistema online de livraria virtual.

3.2. Realizar a garantia de qualidade

O processo de Realizar a Garantia da Qualidade envolve a realização das atividades de qualidade planejadas e sistemáticas descritas no plano de gerenciamento da qualidade, para garantir que o projeto irá empregar todos os processos necessários para atender aos requisitos. Este processo é alimentado pelo plano de gerenciamento da qualidade, que irá guiar todas as atividades de qualidade a serem realizadas. Adicionalmente, métricas e informações sobre desempenho dos trabalhos e outras medições formam parte da entrada do processo. Estas informações são obtidas como resultado do processo de Planejar a qualidade e Realizar o controle da qualidade, respectivamente.

O controle da qualidade envolve a utilização das ferramentas relatadas no planejamento, as quais também serão utilizadas para realizar a garantia da qualidade. Neste contexto, a ferramenta considerada mais importante para a realização da garantia da qualidade é a realização de auditorias.

Uma auditoria consiste em uma revisão estruturada e independente para determinar se as atividades de projeto estão sendo cumpridas como planejado e em consistência com as políticas da organização. As auditorias objetivam oferecer apoio proativo no intuito de melhorar a implementação de processos, através da identificação e compartilhamento de boas práticas, constatadas através das lições aprendidas em projetos similares. Estas ações contribuem na melhoria contínua dos processos e ao consequente aumento na produtividade da equipe e redução de custos.

A auditoria é responsável pelo registro da medição de progresso, que pode ser abrangente, parcial, informal ou formal. Envolve o monitoramento de resultados específicos do projeto a fim de determinar se estão de acordo com os padrões relevantes de qualidade. Como resultado de uma auditoria formal e abrangente deverá ser produzido um relatório contendo as seguintes informações.

- Situação atual do projeto: análise dos custos e prazos dos trabalhos já realizados, da forma mais realista possível.
- Situação futura: previsão do comportamento esperado. Caso exista a possibilidade de algum desvio, o motivo deverá ser registrado.
- Situação das tarefas críticas: situação das tarefas que fazem parte do **caminho crítico**, das tarefas com altos níveis de risco e das tarefas desempenhadas por terceiros.
- Avaliação dos riscos: registro de monitoração dos riscos e suas chances de concretização. Durante a execução podem aparecer novos riscos.

O caminho crítico geralmente envolve a sequência de atividades do cronograma que determina a duração do projeto.

- Informações relevantes a outros projetos: lições aprendidas a partir da realização das auditorias que poderão ser aplicadas a outros projetos.
- Limitações da auditoria: indicar fatores que poderiam limitar a validade da auditoria, premissas que podem ser avaliadas, e se alguém não cooperou ao fornecer informações.

Uma ação preventiva objetiva eliminar a causa de uma potencial não-conformidade ou outra situação potencialmente indesejável.

As auditorias podem ser realizadas por auditores internos à organização executora do projeto ou externos. De forma geral é recomendado que a auditoria seja realizada por uma equipe externa, uma vez que nesse caso, os resultados podem ser mais imparciais e independentes. A escolha depende principalmente dos recursos disponíveis, uma vez que a contratação de auditores externos geralmente envolve um investimento maior. Similarmente a decisão pela execução de auditorias sistemáticas e programadas, em contrapartida de auditorias aleatórias.

Como resultado da execução de auditorias e do processo de realizar a garantia da qualidade de forma geral, a solicitação de mudanças, **ações corretivas**, reparo de defeitos e ações preventivas podem ser levantadas.

Atividades de avaliação

Uma ação corretiva objetiva eliminar a causa de uma não-conformidade ou outra situação indesejável.

1. Qual é o objetivo do processo de realizar a garantia da qualidade?
2. Explique a relação entre os três processos de gerencia da qualidade.
3. Explique de que forma as ferramentas da qualidade podem ser utilizadas para dar suporte ao processo de realizar a garantia da qualidade. Qual você acha mais importante no contexto do processo específico?
4. Explicar com suas palavras a diferencia entre ação corretiva e ação preventiva.
5. O que são as auditorias? Para que servem? Pesquise.

3.3. Realizar o controle da qualidade

Realizar o controle da qualidade envolve o monitoramento e registro dos resultados da execução das atividades de qualidade previstas. O objetivo deste processo é a coleta de informações e dados que possibilitem avaliar o desempenho em relação aos padrões e normas estabelecidos durante o planejamento. A partir dessa análise é possível recomendar mudanças se for necessário.

As atividades de controle da qualidade identificam as causas da baixa qualidade do processo ou produto e recomendam e/ou executam as ações para eliminá-las.

Duas abordagens complementares podem ser adotadas de forma a verificar a qualidade de um produto de software:

- Revisões, onde o software e demais artefatos vinculados ao desenvolvimento do produto são revisados por uma equipe estabelecida para essa finalidade.
- Avaliação automatizada de software, onde o software e demais artefatos vinculados ao desenvolvimento do produto são processados por um programa visando estabelecer a consistência dos mesmos em relação aos padrões adotados.

Prevenção: manter os erros fora do processo, ou seja evitar que eles aconteçam.

Inspeção: manter os erros fora do alcance do cliente, ou seja, podem até acontecer mais devem ser eliminados antes da entrega ao cliente.

O controle da qualidade é realizado de acordo com o plano de gerenciamento da qualidade, levando em conta métricas e demais medidas de desempenho dos trabalhos realizados. Com base em estas informações, os produtos desenvolvidos a serem entregues aos usuários são avaliados. Similarmente as mudanças e reparos que foram solicitados e realizados precisam ser revisados de forma a garantir a corretude e satisfação da qualidade exigida.

Os resultados obtidos a partir das medições são documentados para a sua inclusão no banco de dados de informações históricas do projeto e da organização, de forma geral.

O controle de qualidade garante que as atividades de um programa ocorram conforme planejado. As atividades de controle da qualidade também poderão descobrir falhas no projeto e, assim, indicar mudanças que poderiam melhorar a qualidade.

Atividades de avaliação

1. Qual é o objetivo do processo de realizar o controle da qualidade?
2. Explique de que forma as ferramentas da qualidade podem ser utilizadas para dar suporte ao processo de realizar o controle da qualidade. Qual você acha mais importante no contexto do processo especifico?
3. Determine em cada situação a seguir, quais das atividades são de prevenção e quais de inspeção.
   a) Uma alta ocorrência de defeitos é detectada em componentes de software. A gerência cria uma equipe de qualidade para o planejamento e realização dos testes.

b) Ao fim de cada fase de desenvolvimento são realizadas revisões de documentação e código.

c) Os casos de teste são projetados com base na modelagem dos casos de uso de forma a evitar situações não previstas na hora dos testes de aceitação.

d) Foi detectado o atraso sistemático nas entregas ao cliente. A gerência instaura uma política de padrões de reuso de forma a aumentar a produtividade.

e) São detectadas variações no cronograma. A gerência implanta um processo de controle de mudanças de forma a reduzir os atrasos.

f) É detectado um problema no funcionamento de um dispositivo quando a temperatura atinge 0 grau. É incorporado na bateria de testes para checar o funcionamento a temperatura 0 ou negativa.

4. Sugira situações de prevenção e de inspeção.

4. Ferramentas da qualidade

Os objetivos de qualidade e desempenho são traçados a partir de medidas de produto e de processo sumarizando o desempenho atual do projeto. O gerenciamento quantitativo destas informações permite estabelecer linhas de base para medir e controlar o andamento dos trabalhos nos projetos da organização. Para que isto seja possível, é necessário utilizar ferramentas analíticas para ilustrar o desempenho dos projetos e permitam detectar possíveis variações e determinar suas causas.

Um conjunto de ferramentas estatísticas foi consolidada e associado especificamente ao suporte das atividades vinculadas à gerência da qualidade.

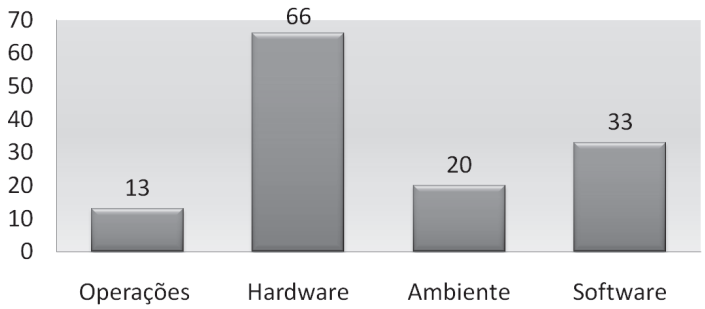
4.1. Histogramas

Na estatística, um histograma é uma representação gráfica da distribuição de frequências de uma massa de medições de variáveis.

O histograma é um gráfico composto por barras verticais (retângulos justapostos) que representam um atributo ou característica de um problema. A base de cada barra corresponde ao intervalo de classe e a sua altura à respectiva frequência. Quando o número de dados aumenta indefinidamente e o intervalo de classe tende a zero, a distribuição de frequência passa para uma distribuição de densidade de probabilidades. A construção de histogramas constitui um estudo preliminar que auxilia na detecção de causas de problemas a partir da distribuição de dados ao longo das barras. Por exemplo, a tabela a seguir apresenta as causas usuais de defeitos em sistemas de computação.

Defeitos	Frequência
Operações	13
Hardware	66
Ambiente	20
Software	33

Os dados em termos absolutos podem ser visualizados graficamente no histograma ao lado.



O histograma permite analisar a frequência de ocorrências para um dado evento, por exemplo, um tipo de falha.

Diagrama de Pareto é um histograma (gráfico de barras) que ordena as frequências das ocorrências, da maior para a menor, permitindo a priorização dos problemas. Mostra ainda a curva de percentagens acumuladas.

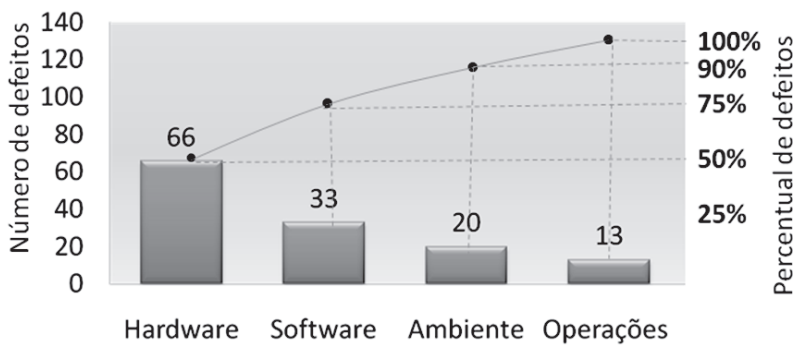
Vilfredo Pareto (Paris, 1848). Político, sociólogo e economista italiano. Pareto introduziu o conceito de ótimo de Pareto, conhecido como Lei de Pareto

A idéia que fundamenta este diagrama é que um grande número de problemas é originado a partir de um pequeno número de causas. Esta regra é chamada de regra 80/20, que sugere que o 80% dos defeitos são originados por 20% das causas.

Para isso, são calculados a frequência relativa (percentual), a acumulada e o correspondente percentual. Desta forma, a tabela dos defeitos apresentada anteriormente fica da seguinte forma:

Defeitos	Freq.	% Freq.	Freq. Acum.	% Freq. Acum.
Hardware	66	50	66	50
Software	33	25	99	75
Ambiente	20	15	119	90
Operações	13	9	132	100

A partir desses cálculos, o Diagrama de Pareto para o exemplo de defeitos é apresentado a seguir.



A maior utilidade do Diagrama de Pareto é a de permitir uma fácil visualização e identificação das causas ou problemas mais importantes ou com maior influência, possibilitando a concentração de esforços sobre os mesmos. Esta análise pode ser de grande utilidade quando os recursos são escassos.

No contexto do exemplo, a partir do diagrama podemos concluir que as áreas prioritárias que requerem uma maior concentração de esforço são as de hardware e software, uma vez que o impacto dos defeitos nestas áreas representa quase o 80% do total.

Atividades de avaliação

1. Defina histograma e diagrama de Pareto estabelecendo claramente a diferencia entre ambos.
2. Explique a regra 80/20 no contexto de causas de defeitos em desenvolvimento de sistemas.
3. Apresente exemplos onde a regra 80/20 se aplica.
4. Defina os conceitos de frequência, percentual de frequência, frequência acumulada e percentual de frequência acumulada
5. Desenhe o histograma para aos seguintes dados e interprete o gráfico resultante.

Mês	# Reuniões
Janeiro	3
Fevereiro	4
Março	6
Abril	9
Maio	11
Junho	4
Julho	3
Agosto	6
Setembro	7
Outubro	6
Novembro	4

6. Desenhe o diagrama de Pareto para aos seguintes dados e interprete o gráfico resultante.

Tipo de Defeitos	#
Modificados	12
Ambiguo	25
Incorreto	30
Incompletos	5

4.2 Diagramas de causa-efeito

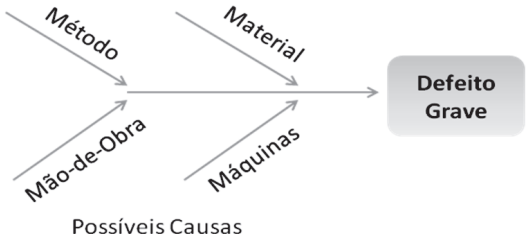
O diagrama de **Ishikawa** ou Espinha-de-peixe é uma ferramenta gráfica utilizada para o gerenciamento e o controle da qualidade (CQ) em processos diversos. A estratégia por trás deste diagrama é que para resolver o problema é preciso conhecer as causas.

O diagrama foi proposto originalmente pelo engenheiro químico Kaoru Ishikawa em 1943 e aperfeiçoado nos anos seguintes. Também é conhecido como: diagrama causa-efeito, diagrama 4M (quatro "emes") e diagrama 6M.

Kaoru Ishikawa (Tokyo 1915 - 1989), foi um engenheiro de controle de qualidade e teórico da administração das companhias japonesas. Ishikawa liderou o movimento pela qualidade total no Japão.

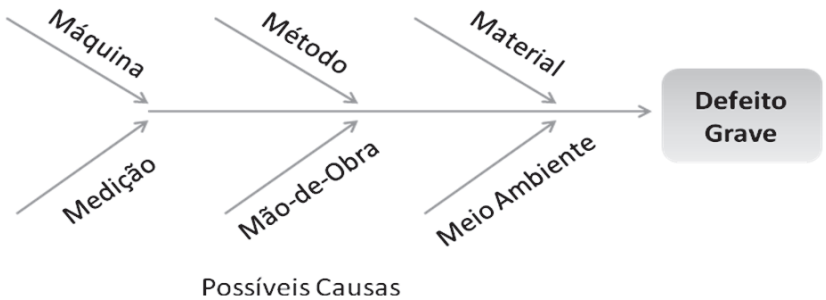
No diagrama 4M são consideradas as seguintes quatro categorias de causas de defeitos:

- Método
- Material
- Mão-de-Obra
- Máquinas



Já em um diagrama 6M as causas podem ser classificadas como obedecendo aos seguintes seis tipos diferentes de causas:

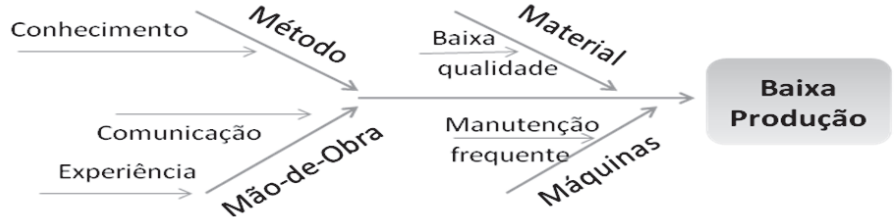
- Método
- Material
- Mão-de-Obra
- Máquinas
- Medição
- Meio Ambiente



A partir de cada categoria, são analisadas as possíveis causas para o defeito sendo analisado. Este sistema permite estruturar hierarquicamente as causas de determinado problema ou oportunidade de melhoria, bem como seus efeitos sobre a qualidade. Permite também estruturar qualquer sistema que necessite de resposta de forma gráfica e sintética.

O diagrama pode evoluir de uma estrutura hierárquica para um diagrama de relações, apresentando uma estrutura mais complexa, na medida em que mais (sub-) espinhos forem adicionados ao diagrama.

Um exemplo de diagrama para o problema de baixa produção em uma fábrica é apresentado a seguir.



A estratégia a ser seguida para a construção do diagrama envolve determinar quais são as operações envolvidas, e a partir daí, determinar, para cada um dos aspectos ou tipos de causas, os problemas específicos que podem estar afetando o desempenho.

Atividades de avaliação

1. Explique o objetivo do diagrama de espinha de peixe e descreva os seus componentes.
2. Desenhe um diagrama espinha de peixe para descrever as possíveis causas dos seguintes defeitos:

   a) Defeitos na especificação dos requisitos de software.

   b) Déficit de atenção de alunos em sala de aula.

   c) Atraso na entrega de pizzas.

   d) Alto índice de defeitos em componente de software.

   e) Insatisfação de pacientes no atendimento de sala de emergência de hospital.

   f) Alto índice de casos de dengue no estado.

   g) Insatisfação por atraso no atendimento dos pedidos de refeição em restaurante.

   h) Dificuldade de manutenção em um sistema de software.

   i) Dificuldade de usabilidade em um aplicativo web.

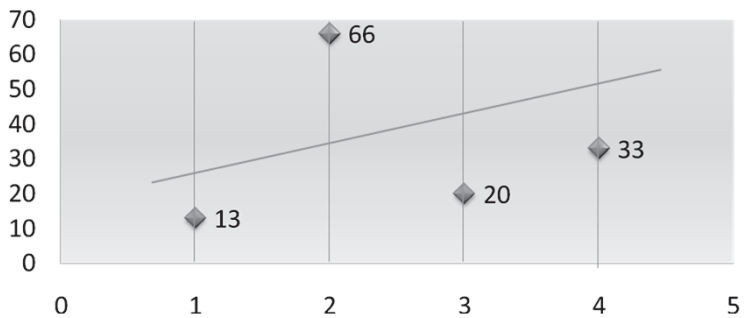
   j) Baixo desempenho de um aplicativo web.

4.3 Gráficos de dispersão.

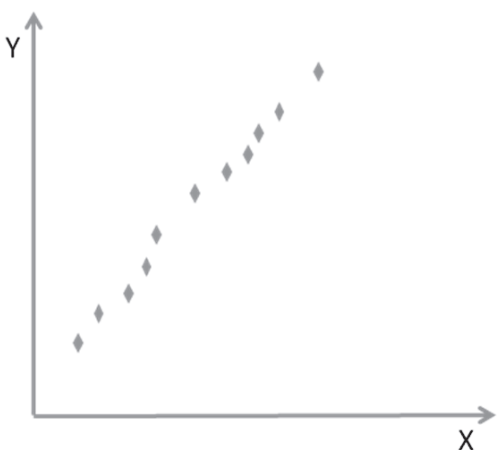
Dados que estejam organizados em colunas ou linhas em uma planilha ou tabela podem ser transportados a um gráfico de dispersão. Um gráfico de dispersão ou scatter chart, constitui a melhor maneira de visualizar a existência de um padrão de relação entre duas variáveis quantitativas, onde uma é definida em função da outra. A construção do gráfico envolve a coleta de dados aos pares de duas variáveis (causa/efeito) para verificar a existência real da relação causal entre essas variáveis em um plano XY. O gráfico é construído sobre um plano onde é mostrado um conjunto de dados numéricos ao longo do eixo horizontal (eixo X) e outro ao longo do eixo vertical (eixo Y). Esses valores são combinados através de pontos de dados únicos os quais são exibidos a intervalos irregulares, ou agrupamentos. Para isso são traçadas as variáveis dependentes versus as variáveis independentes.

A partir da análise do gráfico pode ser determinada a causa raiz de problemas. Quanto mais próximos os pontos estiverem da linha, mais próxima será a relação entre eles. Isto não prova que uma variável afeta a outra, mas torna claro se a relação existe e em que intensidade.

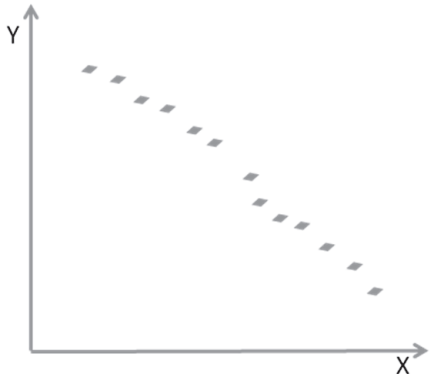
Os gráficos de dispersão são frequentemente usados para exibir e comparar valores numéricos, como dados científicos, estatísticos e de engenharia.



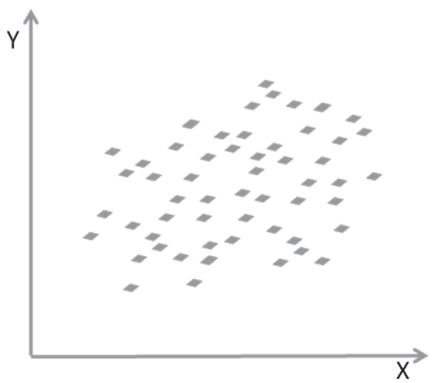
Observando o padrão de disposição dos pontos, é possível concluir sobre a eventual relação entre as duas variáveis. Quando a variável X aumenta implica aumento da variável Y. A partir dessa relação podemos estabelecer que na medida em que uma variável é controlada a outra também é. Esta situação é chamada de correlação positiva. Por exemplo: quantidade de defeitos versus horas extraordinárias.



No caso de correlação negativa, o aumento de X implica na redução de Y. Por exemplo: idade do equipamento versus desempenho.



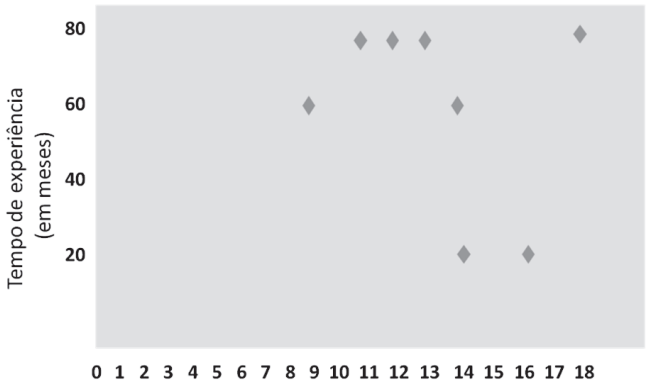
Finalmente, pode se dar o caso em que não seja possível estabelecer uma relação entre X e Y, uma vez que o gráfico da função não pode ser mapeado para uma função puramente crescente ou decrescente.



Por exemplo, considere a seguinte tabela onde é apresentado o número de defeitos encontrados e o tempo de experiência dos inspetores.

Inspetor	Experiência	Defeitos
A	80	13
B	20	14
C	60	09
D	80	18
E	80	11
F	80	12
G	80	11
H	80	12
I	60	14
J	80	13
K	20	16

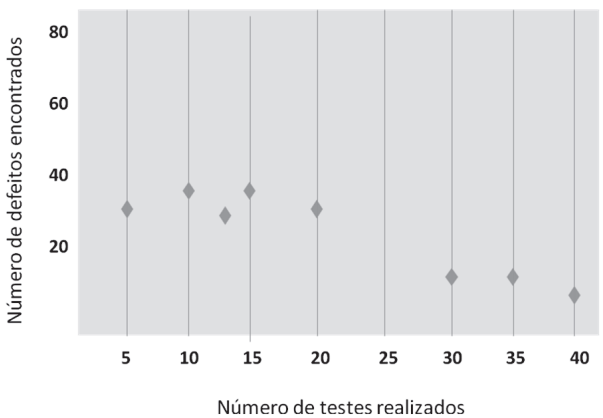
A partir dos dados registrados na tabela surge o seguinte gráfico de dispersão. A partir da análise do diagrama podemos observar que os pontos não apresentam organização linear, sugerindo que o número de defeitos detectados pelos inspetores não estaria correlacionado com a experiência dos mesmos.



Como exemplo de variáveis de natureza quantitativa, ou seja, variáveis que podem ser medidas ou contadas, temos: sinergia, horas de treinamento, intenções, número de horas em ação, jornada, intensidades, velocidade, tamanho do lote, pressão, temperatura, quantidade de defeitos, peso, etc.

Atividades de avaliação

1. Explique as utilidades do gráfico de dispersão.
2. Explique os conceitos de correlação positiva e negativa e seu significado. O que significa que os dados não estão correlacionados?
3. Existe uma clara correlação positiva entre a temperatura e o consumo de cerveja. Forneça 3 exemplos do mundo real onde exista correlação positiva e 3 de correlação negativa.
4. Interprete o gráfico a seguir.



5. Desenhe um gráfico de dispersão para representar a relação entre os seguintes dados:

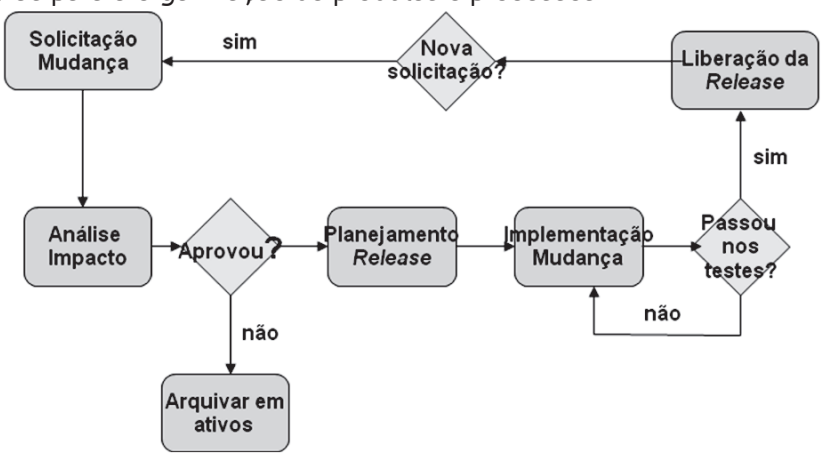
% Assistência às aulas	Nota prova
10	0.5
30	2.5
40	3.5
50	6.0
70	6.8
80	8.5
85	8.0
90	9.4
100	9.8

6. Interprete o gráfico da questão 5. Existe algum tipo de correlação entre os dados?

7. Forneça um exemplo onde os dados não apresentem correlação.

4.4 Fluxogramas

Fluxograma é um tipo de diagrama muito útil no gerenciamento da qualidade utilizado para representar esquematicamente um processo, ilustrando a transição de informação entre os elementos que o compõem. Na prática, o diagrama documenta os passos necessários para a execução de um processo específico e se constitui em uma ferramenta muito utilizada em fábricas e indústrias para a organização de produtos e processos.



De forma geral o retângulo representa uma tarefa ou atividade, o losango denota um estágio de tomada de decisão e as setas (orientadas) indicam a direção do fluxo de informação de uma atividade para outra.

Diversas abordagens e notações podem ser utilizadas para modelar fluxogramas de sistemas computacionais, entre elas o Diagrama de Fluxo de Dados (DFD), diagrama de atividades da UML, etc.

Atividades de avaliação

1. Descreva as principais aplicações do fluxograma.
2. Gere um texto em linguagem natural relatando o fluxo representado no fluxograma acima.
3. Desenhe um fluxograma que modele o fluxo de atividades para trocar um pneu.
4. Desenhe um fluxograma que modele o fluxo de atividades para a construção de uma casa.
5. Modele o fluxo de atividades para a devolução de um item em um pedido de compra.
6. Modele através de um fluxograma as atividades realizadas na retirada de dinheiro de um caixa eletrônico.

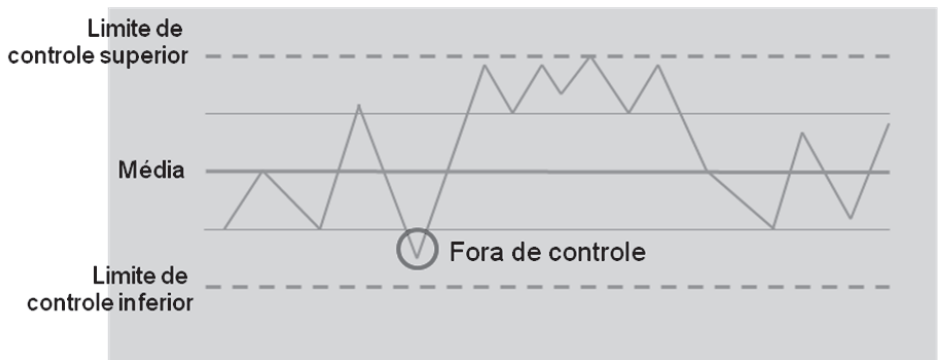
Walter Andrew Shewhart (USA, 1891-1967). Formado em Física, Shewhart é conhecido como o pai do controle estatístico de qualidade por suas valiosas contribuições tanto para a estatística quanto para o controle de qualidade na indústria.

4.5 Cartas de controle

Carta de controle ou gráfico de controle de **Shewhart** é um tipo de gráfico, comumente utilizado para o acompanhamento durante a execução de um processo. A princípio, a partir das amostras extraídas durante o processo, pressupõe-se uma distribuição normal das características da qualidade. No entanto, considerando o princípio da estatística de que todo processo tem variações, o gráfico determina uma faixa chamada de tolerância, limitada pela linha superior (limite superior de controle), uma linha inferior (limite inferior de controle) e uma linha média do processo, cuja variação é determinada estatisticamente.

O objetivo do gráfico é auxiliar a verificar se o processo está sob controle. Tipos de Cartas de Controle:

- Controle por variáveis
- Controle por atributos



O eixo horizontal nos gráficos de controle apresenta o número de amostras, geralmente organizadas no tempo.

O gráfico de controle apresenta três linhas básicas:

- A linha central que fornece a média dos dados processados.
- A linha superior que indica o limite de controle superior (LCS).
- A linha inferior que indica o limite de controle inferior (LCI).

A região determinada entre o limite superior e inferior indica o intervalo aceitável para valores dos dados. Os pontos fora desses limites indicam que o processo se mostra instável.

Por exemplo, o gráfico de controle pode ser utilizado para o controle da ocorrência de defeitos detectados em testes. Flutuações são normais, no entanto, foram formuladas algumas regras que permitem determinar comportamentos suspeitos com maior facilidade no gráfico, tais como:

1. Um ou mais pontos fora dos limites de controle.
2. Dois ou três pontos consecutivos fora dos limites de alerta dois - sigma.
3. Quatro ou cinco pontos consecutivos além dos limites um-sigma.
4. Uma sequência de 7 pontos consecutivos de um mesmo lado da linha central.
5. Seis pontos em uma sequência sempre crescente ou decrescente.
6. Quinze pontos em sequência na zona C (tanto acima quanto abaixo da linha central)
7. Quatorze pontos em sequência alternadamente para cima e para baixo.
8. Oito pontos em sequência de ambos os lados da linha central com nenhum na zona C.
9. Um padrão não usual ou não aleatório nos dados
10. Um ou mais pontos perto de um limite de alerta ou de controle.

No caso em que sete pontos consecutivos caírem fora do limite, de um mesmo lado, caracteriza um problema que precisa ser analisado e resolvido uma vez que configura falta de consistência e previsibilidade do sistema. Esta regra é chamada regra dos sete. Esta situação pode ser observada na figura anterior.

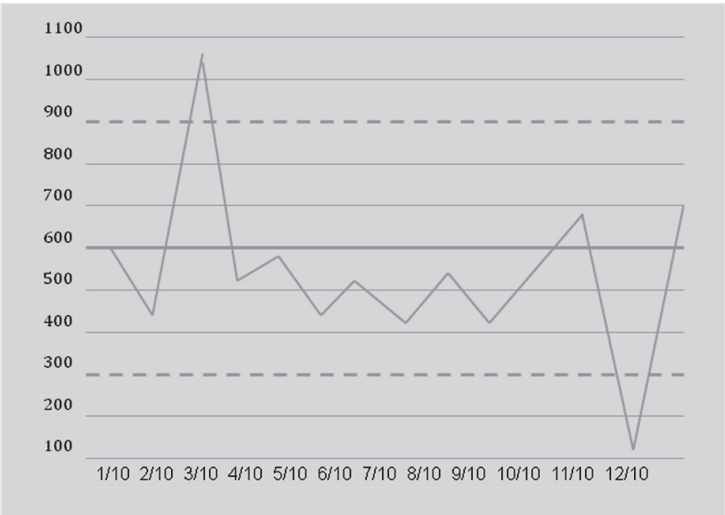
Gráficos de controle podem ser utilizados para monitorar o comportamento e variações de diversas variáveis de projeto, tais como:

- Custo e cronograma.
- Frequência de mudanças;
- Defeitos no projeto.

A partir da análise sobre o gráfico e no caso em que for detectada uma alta variabilidade nos dados, ações corretivas podem ser aplicadas de forma a reduzir essa instabilidade.

Atividades de avaliação

1. Explique a utilidade do gráfico de cartas de controle. Que tipo de dados é representados?
2. Analise o gráfico a seguir e responda:



a) Circule no gráfico o conjunto de pontos que configuram a regra dos sete. O que isso significa?

b) Do seu ponto de vista, este processo se encontra totalmente controlado? Por quê?

c) Determine os valores médio, limite de controle superior e inferior.

4.6 Folhas de verificação

Todas as atividades gerenciais são fortemente baseadas em dados e informação, utilizados como base para o planejamento e o controle das atividades. No caso da gerência da qualidade não é diferente.

Neste contexto, as folhas de verificação são tabelas ou planilhas usadas para facilitar a coleta, organização e análise de dados. O uso de folhas de verificação economiza tempo, eliminando o trabalho de se desenhar figuras ou escrever números repetitivos. Esta organização possibilita a fácil utilização e análise dos dados.

Tipicamente, as folhas de verificação são utilizadas para checar a ocorrência ou não de determinada característica para um conjunto de itens. Os dados são registrados indicando a data, local e responsável pelo registro na folha.

As informações contidas em uma folha podem ser utilizadas para conduzir a elaboração de outros gráficos ou diagramas, como por exemplo, histogramas ou gráficos de controle. Normalmente é utilizada para verificar a frequência com que um problema ocorre, e de que forma se manifesta. A construção de uma folha de verificação segue normalmente as seguintes etapas:

1. Estabelecer a informação a ser estudada;
2. Determinar o tamanho da amostragem, isto é, o período em que será coletada a informação que está sendo estudada;
3. Construir um formulário ou tabela simples e de fácil manuseio;
4. Coletar os dados de forma consistente e honesta.

A folha de verificação apresentada a seguir ilustra um exemplo de registro de ocorrências de diferentes tipos de defeitos ao longo das diversas baterias de testes realizados

Componente	Módulo de controle de usuários			
Data	22/10/2010			
Controlador	João			
Defeitos	Bateria 1	Bateria 2	Bateria 3	Bateria 4
Hardware	IIIII IIII	IIIII I	IIIII III	IIIII IIIII I
Software	IIIII IIIII IIII	IIIII IIIII II	IIIII IIII	IIIII IIIII IIIII III
Ambiente	IIIII	IIII	II	IIIII II
Operações	IIIII I	IIIII	IIIII I	III

Atividades de avaliação

1. Desenhe um exemplo de folha de verificação para o controle do atendimento das diversas especialidades (clínica médica, cardiologia, traumatologia, neurologia) de um posto de saúde ao longo dos dias da semana.
2. Desenvolva outro exemplo de aplicação de folhas de verificação no contexto do mundo real.
3. Desenvolva outro exemplo de aplicação de folhas de verificação no contexto de desenvolvimento de software.

Embora algumas dessas ferramentas já fossem conhecidas nos mais diversos domínios de aplicação, a sua contribuição no contexto de gerência da qualidade é inquestionável. Espera-se que embora nem todos os problemas possam ser tratados por essas ferramentas, ao menos 95% podem ser

satisfatoriamente analisados através delas.

Síntese do capítulo

Nesta unidade é apresentado o conceito de qualidade e seu controle com vistas a atingir a satisfação do cliente. A preocupação com a qualidade se justifica uma vez que influencia em todos os aspectos do desenvolvimento do produto, e principalmente na sua aceitação no mercado.

O estabelecimento de uma cultura na organização a partir da adoção e adequação a normas e padrões nacionais e internacionais torna os processos mais produtivos e eficientes.

Nesta unidade foram abordados os três processos básicos envolvidos na gerencia da qualidade, a saber: planejamento, realizar a garantia e o controle. Finalmente, um conjunto de ferramentas estatísticas para o apoio à realização de tais processos foram apresentadas e ilustradas.

Leituras, filmes e sites

Leituras

ABTN. NBR ISO 9000, Sistemas de gestão da qualidade – Fundamentos e vocabulário. 2000.

ABTN. NBR ISO 9001, Sistemas de gestão da qualidade – Requisitos. 2000.

ABTN. NBR ISO 9004, Sistemas de gestão da qualidade – Diretrizes para a melhoria do desempenho. 2000.

ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO (SOFTEX) MPS.BR- melhoria de processo de software brasileiro: Guia Geral. 2009. Disponível em: http://www.softex.br. Acesso em: novembro 2010.

CAPABILITY MATURITY MODEL INTEGRATION (CMMI) Overview. Disponível em: http://www.sei.cmu.edu/cmmi/. Acesso em: Novembro, 2010.

Site

http://www.abnt.org.br/

Referências

PRESSMAN, R. S. Engenharia de Software. 5ª ed. Rio de Janeiro: McGraw-Hill, 2002.

SOMMERVILLE, I. Engenharia de Software. 6ª ed. São Paulo: Addison Wesley, 2003.

VALERIANO, D. Moderno gerenciamento de projetos. Prentice Hall, 2005.

Capítulo 4

Metodologias e ferramentas da Engenharia de Software

Objetivos

- Nesta unidade são apresentadas as principais metodologias utilizadas para o desenvolvimento de sistemas computacionais, descrevendo o fluxo de atividades, métodos e demais adequações de métodos da Engenharia de Software para cada caso.

1. Introdução

No inicio do livro, a Engenharia de Software foi definida como uma disciplina que utiliza um conjunto de métodos, técnicas e ferramentas para analisar, projetar e gerenciar o desenvolvimento e manutenção de software, visando produzir e manter software dentro de prazos, custos e qualidade estimados.

A ausência de um processo disciplinado para o desenvolvimento de sistemas implica em uma preocupação crescente de que sérios problemas no desenvolvimento, implantação e manutenção do sistema podem ser ocasionados. No entanto, nem sempre é possível aplicar as abordagens convencionais da engenharia de software, seus princípios, conceitos e métodos para o desenvolvimento de qualquer tipo de aplicação. Muitos deles podem, mas sua aplicação pode exigir uma abordagem diferente.

Um maior sucesso no desenvolvimento e utilização de sistemas complexos e de grande porte promove uma necessidade premente de abordagens disciplinadas de engenharia, assim como novos métodos e ferramentas para o desenvolvimento de tais sistemas, os quais devem levar em conta as características específicas do novo domínio, ambientes e cenários operacionais e tecnológicos, que adicionam desafios ao desenvolvimento de novas aplicações.

Neste contexto, a aplicação de princípios científicos sólidos, de engenharia e de gestão, e abordagens disciplinadas e sistemáticas se tornam indispensáveis para o bem sucedido desenvolvimento, implantação e manutenção de sistemas e aplicações de alta qualidade.

Os fundamentos científicos para a engenharia de software envolvem o uso de modelos abstratos e precisos que permitem ao engenheiro especificar, projetar, implementar e manter sistemas de software, avaliando e garantindo

sua qualidade. Além disso, a engenharia de software deve oferecer mecanismos para se planejar e gerenciar o processo de desenvolvimento de um sistema computacional.

Uma metodologia envolve princípios filosóficos que guiam uma gama de métodos que utilizam ferramentas e práticas diferenciadas para realizar algo. Um método da engenharia de software consiste em uma abordagem estruturada para o desenvolvimento de software de alta qualidade, e que geralmente é aplicado no contexto de uma metodologia de desenvolvimento.

2. Metodologia Estruturada

A **metodologia estruturada** de desenvolvimento de software foi desenvolvida inicialmente na década de 1970, e tem seu foco na identificação dos componentes funcionais básicos do sistema.

Na visão de desenvolvimento da metodologia estruturada, o sistema é visualizado no modelo entrada-processo-saída, onde os dados são considerados separadamente das funções. São estas últimas as responsáveis pela transformação de dados de entrada em dados de saída.

Na metodologia estruturada existe uma clara separação entre funções e dados, onde as primeiras são ativas e implementam a lógica da aplicação, enquanto os dados são entidades de informação passivas, normalmente estruturados em repositórios.

A abordagem estruturada orientada a função apresenta alguns problemas em relação a possíveis redundâncias e inconsistências, tornando as funções difíceis de serem integradas e reutilizadas. Adicionalmente, a dependência das funções em relação aos dados torna o sistema difícil de ser mantido, uma vez que é preciso conhecer como os dados se encontram armazenados e estruturados. Esta dependência faz com que eventuais mudanças na estruturação dos dados impactem nas funções que os utilizam, precisando também passar por manutenção.

O desenvolvimento utilizando a abordagem estruturada inicia a partir da elaboração do chamado **anteprojeto**, que objetiva identificar o tipo de serviço de processamento de dados, objetivos e demais recursos necessários. Esta fase conta com a ativa participação do usuário e aborda as seguintes atividades:

- Identificação de objetivos global e específicos do software.
- Definição de abrangência e as macro-funções existentes, assim como os envolvidos.
- Análise de dados visando identificar as principais entidades, seus atributos e relacionamentos.
- Análise de funções envolvendo a análise de problemas e possíveis soluções.

Como resultado desta fase é gerado o modelo preliminar de contexto, descrevendo as macro-funções e seus fluxos de entrada e saída, assim como os repositórios de dados, levando em conta as entidades que fazem parte do modelo lógico de dados.

Adicionalmente, alternativas de hardware e software de apoio precisam ser avaliadas, assim como também a realização de estimativas de recursos e prazos.

Uma vez aprovado o anteprojeto, dá inicio o desenvolvimento do **projeto lógico**, envolvendo a especificação detalhada dos elementos de software a nível lógico, assim como também dos procedimentos externos ao computador, tais como as informações de entrada e sua distribuição na saída.

- Modelagem de dados de forma a tornar o modelo de dados normalizado (sem redundâncias lógicas) e devidamente documentado através do dicionário de dados.
- Modelagem de processos envolve o refinamento sucessivo da documentação gerada no anteprojeto, particionando os processos e sub-processos em funções, até níveis mais refinados de detalhe.
- Definição de entradas e saídas que irão alimentar e ser geradas pelos processos, tais como formulários, relatórios, telas, etc.

A partir do projeto lógico, o **projeto físico** objetiva detalhar os elementos de software a nível físico.

- Projeto físico do banco de dados envolve a organização das entidades e seus atributos visando o melhor desempenho e outros aspectos de qualidade do software.
- Projeto da estrutura do software descrevendo a estrutura hierárquica dos módulos e as informações trocadas entre eles.

Finalmente o sistema é implementado em uma linguagem de programação (estruturada) e as devidas simulações e testes realizados.

A principal ferramenta utilizada para modelar o funcionamento do sistema na Análise Estruturada é o Diagrama de Fluxos de Dados ou DFD. O DFD consiste em uma rede de processos inter-relacionados utilizado para ajudar a particionar e organizar os requisitos do sistema, mostrando a decomposição das principais funções e todas as interfaces dos vários módulos.

Adicionalmente, o Dicionário de Dados (DD) é utilizado como auxilio para lidar com a enorme quantidade de detalhes e documentando cada um dos fluxos de interface e depósitos de dados em qualquer DFD. Idealmente, cada um dos processos dos DFDs precisam ser documentados de um modo rigoroso utilizando ferramentas superiores ao texto narrativo, tais como:

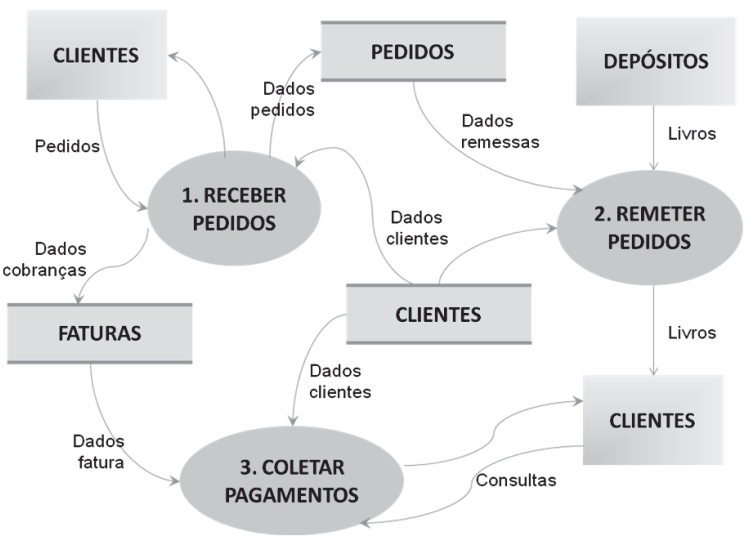
- Português estruturado
- Tabelas de Decisão
- Árvores de Decisão

O diagrama de fluxo de dados DFD descreve o fluxo de informação e as transformações aplicadas na medida em que os dados se movimentam da entrada para a saída. Um DFD envolve os seguintes componentes:

Elemento	Notação gráfica
O processo representa uma entidade transformadora de informações que faz parte do sistema sendo modelado.	Process
Uma entidade externa representa um produtor ou consumidor de informação que reside fora do sistema.	Entidade Externa
O fluxo de dados representa o deslocamento de um item de informação ou um conjunto.	→
O depósito de dados representa um repositório de dados armazenados para seu uso pelos processos.	Repositório de dados

As diretrizes básicas para a construção de um DFD são as seguintes:

1. Escolher nomes significativos para os processos, fluxos, depósitos e terminadores.
2. Numerar os processos.
3. Refazer o DFD tantas vezes quantas forem necessárias até obter uma boa estética.
4. Evitar DFD complexos demais.
5. Certificar-se de que o DFD seja internamente consistente além de manter a consistência com os outros DFD.



O Dicionário de Dados (DD) é um repositório de dados sobre dados que contém as definições rigorosas de todos os elementos que compõem o DFD, tais como:

- Fluxos de dados;
- Componentes de fluxos de dados;
- Arquivos;
- Processos.

Como exemplo de Dicionário de Dados (DD) para os repositórios no exemplo acima temos:

Clientes = Nome_Cliente + CPF + Endereço_cliente +

Factura = Número_factura + Número_conta_cliente +

Total_pagamento + ID_vendedor

O conjunto do DFD com o DD compõe a especificação mínima do sistema em análise.

1. Pesquise em livros e apostilhas e escolha um exemplo de desenvolvimento seguindo a metodologia estruturada o suficientemente completo. Recreie os diagramas.

3. Metodologia Orientada a Objetos

Na década de 1980 a abordagem estruturada deu lugar a métodos orientados a objetos. A abordagem **orientada a objetos** (OO) surge em resposta ao constante aumento na complexidade dos sistemas dificilmente gerenciada através de metodologias estruturadas.

Na abordagem orientada a objetos o mundo real é composto por objetos (entidades) os quais encapsulam a sua estrutura de dados (estado) junto ao seu comportamento funcional. Neste contexto, um sistema é modelado como um conjunto de objetos interagindo através da troca de mensagens. Esta forma de descrever os sistemas resulta mais intuitiva não somente para os profissionais, mas também para os usuários e clientes, facilitando o entendimento e a comunicação entre os envolvidos. Benefícios da utilização da abordagem orientada a objetos têm sido largamente explorados na literatura, entre eles temos:

- Redução do gap semântico entre o domínio do problema e da solução computacional, facilitando o entendimento e agilizando a troca de informações.
- Capacidade de modelar uma ampla variedade de problemas.
- Facilidade de reuso.
- Facilidade de manutenção, entre outras.

Com a popularização da orientação a objetos, inúmeros métodos e técnicas surgiram no intuito de dar apoio às diversas atividades de desenvolvimento. No entanto, houve um esforço conjunto de um consórcio de empresas em parceria com pesquisadores e engenheiros de software na procura por métodos e técnicas de desenvolvimento padrão. Como resultado, a linguagem UML (Unified Modeling Language) foi estabelecida em 1997 pela OMG (Object Management Group), como linguagem padrão para a modelagem de sistemas orientados a objetos, colocando um ponto final à chamada guerra dos métodos.

É importante salientar que a UML é uma linguagem que pode ser utilizada para modelar qualquer tipo de sistema, embora se trate de um padrão para OO. Isto significa que a UML não prescreve nenhum procedimento para sua utilização, ou seja, que a princípio, a sua utilização não é restrita ao contexto de uma metodologia específica. Consequentemente, a UML pode ser adotada no contexto de qualquer metodologia sendo utilizada para o desenvolvimento de um sistema.

O modelo de desenvolvimento desenhado para atender às necessidades do desenvolvimento orientado a objetos é o Processo Unificado. Este modelo estabelece a execução dos diversos fluxos de atividades acontecendo ao longo de fases específicas.

Fluxo de requisitos onde acontece a especificação dos requisitos funcionais através de diagramas de casos de uso sob a perspectiva de utilização dos clientes. Este fluxo tem seu ápice na **fase de concepção,** mas continua ainda durante a fase de elaboração. Durante esta fase, o fluxo de requisitos é responsável pela captura do 80% dos requisitos e entre o 5 e 10% da implementação, sendo que o produto gerado como resultado é o chamado Modelo de Casos de Uso

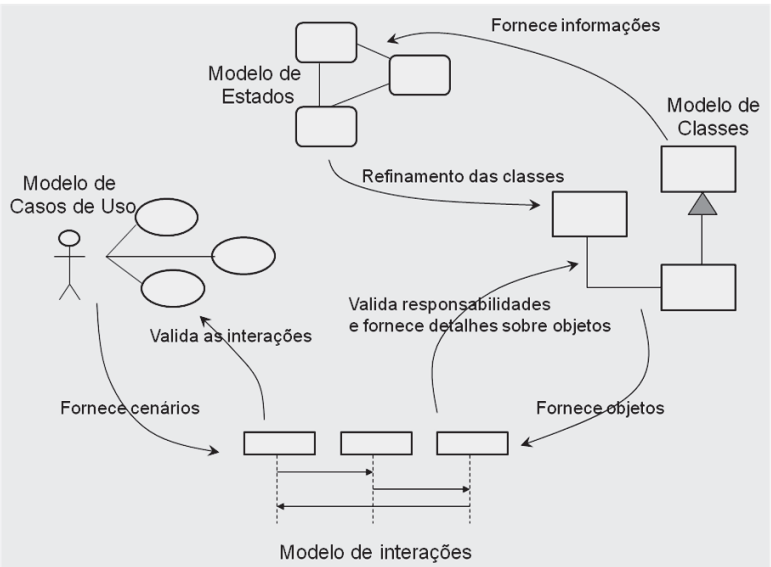
Fluxo de análise consiste no refinamento dos requisitos especificados no fluxo anterior, envolvendo a construção de diagramas de classes conceituais, diagramas de interação e estados. Este fluxo tem seu ápice na **fase de elaboração**. A cada iteração um conjunto de casos de uso é realizado e adicionado ao produto da iteração previa. Desta forma, o modelo cresce na medida em que mais casos de uso são analisados. Como resultado do fluxo é gerado o Modelo de Análise.

O detalhamento dos casos de uso contribui na identificação de novas classes e/ou a reutilização de classes existentes através dos mecanismos propiciados pela orientação a objetos, como a herança.

O fluxo de projeto envolve a realização dos casos de uso levando em conta detalhes de implementação. Os artefatos são organizados em subsistemas com interfaces bem definidas. O produto gerado como resultado é o Modelo de Projeto.

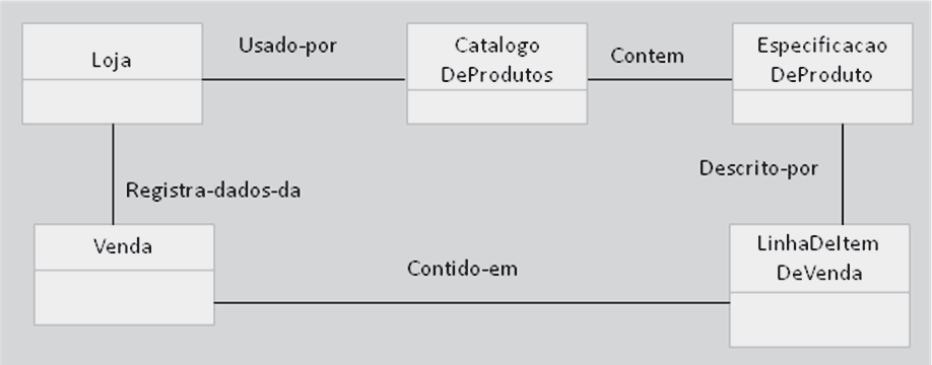
No Fluxo de implementação é realizada a implementação do sistema em termos de componentes e tem maior importância na **fase de construção**. Este fluxo envolve a tradução das decisões de projeto para uma linguagem de programação orientada a objeto específica, tal como Java ou C++. Finalmente, no Fluxo de testes os componentes executáveis são testados para então serem disponibilizados a usuários finais.

Exemplos de artefatos de modelagem gerados no processo de desenvolvimento de sistemas orientados a objetos foram apresentados ao longo das unidades. De forma geral, o conhecimento sobre o sistema sob desenvolvimento é construído ao longo das iterações no processo de desenvolvimento. Os artefatos de modelagem gerados como resultado de cada fase servem de entrada para a geração de novos artefatos, os quais por sua vez vão passando por sucessivos refinamentos, como ilustrado na figura a seguir.

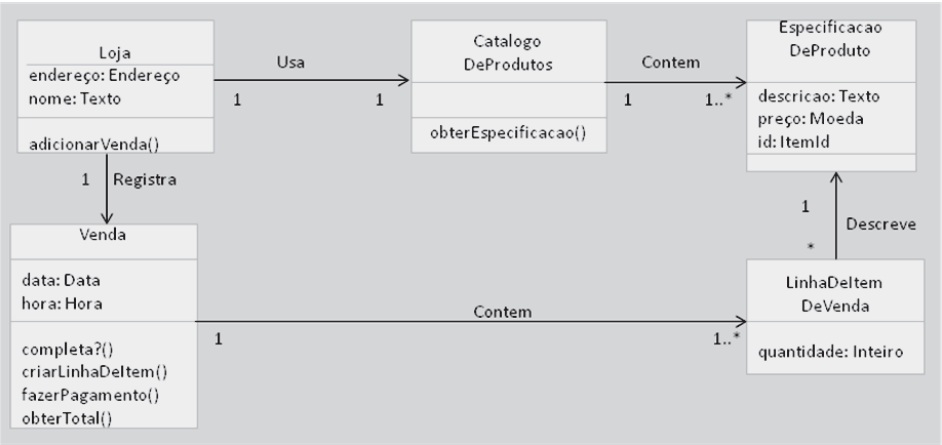


A UML apóia o desenvolvimento incremental onde, ao longo das iterações, os artefatos de projeto são refinados até atingir o nível de detalhamento e rigor adequados, de forma que as fases subsequentes possam ser realizadas. Uma vantagem da utilização da UML é que a mesma notação pode ser utilizada ao longo das diferentes fases e fluxos.

Como exemplo, considere o problema do desenvolvimento de um sistema de controle de vendas em uma mercearia. O diagrama a seguir apresenta a visão parcial de um diagrama de classes no nível de especificação para o problema.



Na medida em que novas informações são incorporadas no processo, assim como a elaboração de outros diagramas complementares, como por exemplo, os diagramas de interação, o diagrama de classes evolui no nível de detalhes. A figura a seguir apresenta um diagrama de classes de análise, onde os atributos e alguns métodos foram incorporados, assim como a multiplicidade nos relacionamentos.



Já em uma versão de projeto, o diagrama é completado em relação aos aspectos comportamentais das classes, e outros aspectos, tais como visibilidade de atributos e métodos são considerados.

Métodos orientados a objetos, de forma geral, se mostram mais adequados para o desenvolvimento de sistemas interativos, por exemplo. Neste sentido, não existe um método ideal que possa ser aplicado às diferentes áreas com o mesmo nível de sucesso e adequação. Isto é, não existe a bala de prata no contexto dos métodos e metodologias.

No caso de **sistemas críticos**, a especificação e desenvolvimento dos sistemas devem ser guiados pelas dimensões de confiança do sistema, tais como: riscos, segurança, proteção, confiabilidade e demais fatores cruciais para tornar o sistema adequado à sua especificação.

O desenvolvimento de **aplicações web** também envolve certas particularidades inerentes às características do domínio específico. Na sua maioria, no desenvolvimento de aplicações web são considerados mais prioritários os seguintes atributos: foco em redes e concorrência, desempenho sob estresse, voltados a dados, imediatismo, segurança e estética. Consequentemente, o desenvolvimento deste tipo de sistemas necessariamente deve incluir, entre outras, a utilização de técnicas de prototipagem e maior ênfase na realização de testes de carga e estresse, por exemplo.

Similarmente, o desenvolvimento de aplicações em outros domínios tais como TV Digital, sistemas autônomos, tempo real e embarcados em dispositivos móveis, etc., pode requerer de abordagens especificamente adaptadas da Engenharia de Software de forma a melhor atender aos requisitos e restrições inerentes.

Os sistemas críticos são sistemas técnicos ou sociotécnicos dos quais as pessoas ou negócios dependem, e onde uma falha pode resultar em perdas econômicas significativas, danos físicos ou ameaça à vida. Portanto, falhas neste tipo de sistemas geralmente não são toleráveis.

Atividades de avaliação

1. A partir do exemplo escolhido e modelado utilizando a abordagem estruturada no exercício anterior, modele outra solução utilizando a abordagem orientada a objetos.
2. Descreva qual o foco de cada uma das metodologias apresentadas.
3. Estabeleça um comparativo entre as metodologias apresentadas.
4. Aprofunde na pesquisa sobre as características dos sistemas críticos e para web e determine quais adequações nos métodos da Engenharia de Software tradicional precisam ser realizadas para atender de forma eficaz à sua especificação.

Leituras, filmes e sites

Nesta unidade foram apresentadas metodologias de desenvolvimento de sistemas mais amplamente utilizadas: a metodologia estruturada e a orientada a objetos. Para cada uma delas foram estabelecidas as principais características, fases, métodos e artefatos envolvidos na modelagem.

Leituras

KENDALL, Scott. O Processo Unificado Explicado. Bookman, 2003.

Booch, G.; Rumbaugh, J.; Jacobson, I. UML - Guia do Usuário. Campus, 2000.

FURLAN, J. D. Modelagem de Objetos Através da UML; Makron Books, 1998.

JACOBSON, I. Object-Oriented Software Engineering, Addison-Wesley,1992.

KRUCHTEN, P. The Rational Unified Process: An Introduction, ObjectTechnology Series, Addison-Wesley, 1998.

Referências

Booch, G. Object-Oriented Analysis and Design with Applications, 2nd edition, Benjamin/Cummings Publishing Company, Inc, 1994.

COAD, P.; YOURDON, E. Análise Baseada em Objetos, Editora Campus, 1992.

POMPILHO, S. Pompilho. Análise Essencial: Guia Prático de Análise de Sistemas.IBPI Press, Editora Infobook, Rio de Janeiro, 1995.

PRESSMAN, R. S. Engenharia de Software. 5ª ed. Rio de Janeiro: McGraw-Hill, 2002.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI W.; EDDY, F.; LORENSEN, W. Modelagem e Projetos Baseados em Objetos, EditoraCampus, 1994.

SOMMERVILLE, I. Engenharia de Software. 6ª ed. São Paulo: Addison Wesley, 2003.

YOURDON, E. Object-Oriented Systems Design: an Integrated Approach, Yourdon Press Computing Series, Prentice Hall, 1994.

Sobre a autora

Mariela Inés Cortés é doutora em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (2003) e mestre em Sistemas de Computação pelo Instituto Militar de Engenharia do Rio de Janeiro (1999). Sua alma mater é a Universidade Nacional de La Plata, onde completou os estudos de graduação em Ciências da Computação. Especialista na área de Engenharia de Software, atualmente é professora adjunta na Universidade Estadual do Ceará, vinculada ao Curso de Ciências da Computação e com uma ativa participação no Mestrado Acadêmico. Adicionalmente, coordena o Laboratório de Qualidade e Padrões de Software (LAPAQ) e lidera o Grupo de Engenharia de Software e Sistemas Inteligentes (GESSI).